



UPPSALA
UNIVERSITET

UPTEC F14 002

Examensarbete 30 hp
Januari 2014

Refinement of an emulator for the physical layer of the wireless communication of sensor networks

Emil Eriksson



UPPSALA
UNIVERSITET

**Teknisk- naturvetenskaplig fakultet
UTH-enheten**

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
<http://www.teknat.uu.se/student>

Abstract

Refinement of an emulator for the physical layer of the wireless communication of sensor networks

Emil Eriksson

Wireless sensor networks have applications in many different fields and environments, each with their own set of challenges to be overcome. If we can measure and compensate for the properties of the wireless environment at an early stage of the deployment, we will be able to have the sensor network operational in a smaller time frame.

This report describes the development of an emulator for wireless sensor network of up to eight 802.15.4 compliant sensor nodes. The FPGA-based emulator has been enhanced with several new features to remove some shortcomings of the existing emulator. The emulator now provides the ability to import previously recorded data representing time varying gains of wireless links, the ability to store the signal data generated by the sensor node for later review, as well as a more intuitive user interface to make the emulator more useful as a research tool. A latency issue that previously prevented the use of any communications protocol depending on transmission acknowledgments has been studied and a solution to the issue is presented. A way of introducing external interference to the emulated system has also been investigated and provides a straightforward way of introducing interference to the system without modifying the existing hardware and software. Most added features have been designed to be easily modified or expanded upon.

Handledare: Tomas Olofsson
Ämnesgranskare: Anders Ahlén
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTec F14 002

Refinement of an emulator for the physical layer
of the wireless communication of sensor
networks

Emil Eriksson

Abstract

Wireless sensor networks have applications in many different fields and environments, each with their own set of challenges to be overcome. If we can measure and compensate for the properties of the wireless environment at an early stage of the deployment, we will be able to have the sensor network operational in a smaller time frame.

This report describes the development of an emulator for wireless sensor network of up to eight 802.15.4 compliant sensor nodes. The FPGA-based emulator has been enhanced with several new features to remove some shortcomings of the existing emulator. The emulator now provides the ability to import previously recorded data representing time varying gains of wireless links, the ability to store the signal data generated by the sensor node for later review, as well as a more intuitive user interface to make the emulator more useful as a research tool. A latency issue that previously prevented the use of any communications protocol depending on transmission acknowledgments has been studied and a solution to the issue is presented. A way of introducing external interference to the emulated system has also been investigated and provides a straightforward way of introducing interference to the system without modifying the existing hardware and software.

Most added features have been designed to be easily modified or expanded upon.

Contents

1	Introduction	9
2	Related work and contributions	12
2.1	Related work	12
2.2	Contributions	12
3	System overview	14
3.1	Sensor nodes	15
3.1.1	Over-the-air transmissions	18
3.2	Peripheral equipment	19
3.3	Emulator	21
3.3.1	Emulator Components	21
3.3.2	Transmitting through the emulator	23
3.4	Software tools	25
3.4.1	LabVIEW	25
3.4.2	Contiki	25
3.4.3	Cooja	26
4	Modifying the LabVIEW host interface	27
4.1	Importing channel data	27
4.2	Exporting sensor signal data	29
4.3	General modifications	29
4.3.1	Signal monitoring	29
4.3.2	Channel monitoring	30
4.3.3	Channel information	30
4.3.4	Status	30
4.3.5	Hardware setup	30
4.3.6	Errors	30
4.4	Verification of the host interfaces usability	31
4.4.1	Importing channel data	31
4.4.2	Exporting signal data	31
4.4.3	General modifications	31
5	Latency	33
5.1	Verifying the latency	33
5.2	Adding latency tolerance to the system	34
5.2.1	Altering the MAC protocol	34
6	Adding interference/noise sources	38
6.1	Establishing a baseline PRR	38
6.2	Connecting a sensor node as a noise source	39
7	Summary and conclusions	41
7.1	Host emulator interface	41
7.1.1	Channel data import	41
7.1.2	Export of resulting data	41
7.2	Latency	41
7.3	Interference sources	42
7.4	Further work	42
7.4.1	Further testing and development	42
A	Emulator description	46
A.1	User manual	46
A.1.1	Hardware setup	46
A.1.2	Required software	49
A.1.3	Configuring the emulator	51
A.1.4	Initializing the sensor nodes	54
A.1.5	Stopping execution	56
A.2	Emulator Host Controls and Indicators	56
A.2.1	Signal Monitoring	56
A.2.2	Channel Monitoring	58
A.2.3	Channel Information	61
A.2.4	Status	62
A.2.5	Hardware Setup	65
A.2.6	Errors	67

B The Channel Data Import subVI	69
C The Signal Data Export subVI	71

List of Abbreviations

ADC	Analog to Digital Converter
ASIC	Application-Specific Integrated Circuit
BER	Bit Error Rate
DAC	Digital to Analog Converter
FPGA	Field-Programmable Gate Array
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LabVIEW	Laboratory Virtual Instrumentation Engineering Workbench
MAC	Media Access layer
MCU	Microcontroller Unit
NI	National Instruments
PC	Personal Computer
PCIe	Peripheral Component Interconnect Express
PRR	Packet Reception Ratio
RDC	Radio Duty Cycle
RF	Radio Frequency
RFSA	Radio Frequency Signal Analyzer
RFSG	Radio Frequency Signal Generator
RSSI	Received Signal Strength Indication
SINR	Signal to Interference and Noise Ratio
UART	Universal Asynchronous Receiver/Transmitter
VI	Virtual Instrument

List of Figures

1	Signal and data monitoring flow of the system setup. Via separate interfaces, the PC sets up the behavior of the emulator as well as any sensor nodes connected to it. Both the emulator and the individual nodes are able to relay control information back to the PC. Both the input signals to, and the output signals from the emulator are combined into a single signal by the combiners. The individual signals are extracted by tuning a receiver to the carrier frequency and bandwidth of the desired signal.	15
2	The Zolertia Z1 sensor node uses the IEEE 802.15.4 compliant radio transceiver CC2420.	16
3	A close up view of the Zolertia Z1's RF components. Note especially the capacitor marked as C62, here soldered to provide the output signal on an external antenna via the U.FL connector. . .	17
4	The back-to-back connection of the power splitters/combiners allows the simultaneous connection of up to eight sensor nodes to both the input and output connectors of the emulator.	20
5	The internal signal and data monitoring flows of the emulator. The transmissions are converted and sampled by the signal analyzer (RFSA), after which the sampled signal is processed by the FPGA before being output again by the signal generator (RFSG). . .	21
6	The resulting PRR based on different waiting times for acknowledgement reception. Note the periodical decrease in PRR that occurs every 0.3-0.4 ms	35
7	The behavior of two nodes, A and B, as they attempt to complete a packet transmission under different conditions. The upper case is over-the-air transmission, the middle case is transmission through the emulator without the modified protocols, and the bottom case is through the emulator while using the modified protocols. Note that relative sizes of the different blocks are exaggerated to illustrate the nodes transmission behavior and do not accurately reflect the relative times it takes to perform the operations.	36
8	The results of the second measurement of PRR, using a higher resolution scale compared to the first test.	37
9	A comparison between theoretical and measured PRR. The two curves are similar, although the theoretical values seem to have a much sharper rise as SINR increases.	39
10	The ExpressCard controller interface connects the emulator and your PC.	46
11	Make sure to connect the power splitters/combiners as pictured.	47
12	Connect the power splitter/combiner to the emulators input and output ports.	48
13	Each sensor node must be connected to a port on the 8/1 power combiner/splitter.	48

14	Use the Measurement and Automation Explorer to find installed software from National Instruments.	50
15	In the Devices and Interfaces part of the Measurement & Automation Explorer you should be able to identify the hardware components of the emulator.	52
16	Default values of the remaining hardware setup configuration options.	53
17	Use the right click dialog to change the default values of your controls.	55
18	The reset button of the Zolertia Z1 sensor node. Sensor node appearance and circuit board layout may differ between manufacturers.	56
19	The signal monitoring tab of the emulator.	57
20	The channel monitoring tab of the emulator.	59
21	The channel information tab of the emulator.	61
22	The status tab of the emulator.	63
23	The hardware setup tab of the emulator.	65
24	The errors tab of the emulator.	67
25	The channel data import subVI iterates through the data fields of a specified file and outputs that data to a higher level VI. . .	70
26	The signal data export subVI continuously appends data to a specified file.	72

List of Tables

1	CC2420 radio transceiver specifications.	16
2	Zolertia Z1 specifications.	17
3	The assignment of carrier frequencies for each of the eight sensor nodes attached to the emulator. Each frequency channel is 5 MHz in bandwidth. Each of these channels are represented by a number specified by the IEEE 802.15.4 standard.	24
4	The channel data file structure. The initial fields are for site and test information, followed by an array of structure arrays, each element containing the information of a specific channel.	28
5	PRRs achieved during transmission tests where the only interference source is the background interference present.	38

1 Introduction

Today, wireless networks are everywhere. The technology has evolved rapidly over the last few decades to a point where a wireless transceiver is a cheap and compact component that can be integrated in a wide variety of devices. One type of device that makes great use of wireless communication is wireless sensors. Since wireless, self powered sensors are mobile they can be used for a wide variety of applications where a wired sensor network would be problematic or even impossible to deploy. For example, wireless sensor networks can be used to track the movement and interactions of animals in the wild or to monitor soldiers on a battlefield. Installation of wireless network components is also much less intrusive than wired sensor networks which can be desirable in historical buildings or at other sites where there is a need for the sensor network to provide minimal disturbance.

Wired and wireless networks provide different benefits and drawbacks and may be preferable in different situations and settings. There are some areas where both wired and wireless sensor networks are viable alternatives but, historically, wired sensor networks have been favored, like local area networks in office environments. The main reasons for this are that wired sensor networks are generally less susceptible to electromagnetic interference that may be caused by nearby equipment, when compared to their wireless counterpart. Wired sensor networks also offer bandwidths that are around an order of magnitude greater than what can be achieved in wireless sensor networks. As a result, wired sensor networks have long been the norm. However, using wireless sensor networks rather than traditional wired sensors will greatly increase the flexibility of the sensor network, especially when considering the issues of network topology, and the deployment, reconfiguration, and maintenance of the physical network. Provided that the overall performance of the sensor network is still adequate for a given application, these advantages could be a great benefit to the network. For example, most modern, industrial facilities use control engineering to ensure consistent quality of products or to minimize waste of materials and energy. In these settings, networks of sensors are used to gather information on the state of the processes by continuously monitor temperatures, pressures, flows, and a myriad of other parameters. The reliability of these sensors, in terms of both the gathering and relaying of data, is imperative for the control process to function efficiently. Because of the high risk of interference in these settings, wired sensors are often used. However, if it is possible to measure and compensate for the wireless characteristics of the site, great benefits could come from choosing a wireless sensor network.

Many well known techniques for handling the drawbacks of wireless networks exist but to determine the feasibility of using a wireless sensor network in a specific setting, field tests are necessary to become familiar with the characteristics of the available wireless channels and to realize the consequences these characteristics have on communication performance. However, getting physical access to a site and thoroughly measuring the characteristics of the available wireless channels can be both time consuming and problematic.

Theoretical knowledge about the effect of various aspects of wireless communications on the performance of the wireless network is ample. However, the complete interaction of all these aspects, as well as the large scale effects of having several nodes interacting with each other, is difficult to recreate reliably even when using advanced simulation tools.

An alternative to simulation is to create an emulation environment. While a simulation models the behavior of the system to some degree of accuracy, an emulation mimics the outward behavior of the system completely. The topic of physical layer emulation has been investigated previously. A large scale setup of a physical layer emulator was created by The Emulator Team at Carnegie Mellon university. Their findings showed that the performance measured in an emulated environment is more easily validated than the performance of a simulated environment, and that the emulated environments aids the development and evaluation of enhanced wireless protocols [1]. Another project at Uppsala university by Danielsson and Ågren created a proof-of-concept version of an emulator for the physical layer of the wireless communication of sensor networks [2]. This report can be seen as an extension to the work done by Danielsson and Ågren.

The goal of this project is to create a physical layer emulator that can be used in the following process:

- Measure the characteristics of the available wireless channels,
- save the gathered data,
- load gathered data onto a piece of equipment that can emulate the physical layer of the wireless communication in real time and,
- directly connect the antennae of physical sensor nodes to the emulator.

This process would allow us to perform repeatable tests in a laboratory environment, using the same type of hardware that would be used for the deployed sensor network. Using this emulator to perform accurate early testing will provide the ability to detect and address issues with the network before deploying the equipment to the site. This approach would also be beneficial because it could save much time, money, and effort.

This report details the work done to improve certain issues of an existing emulator of this type.

The report is organized as follows. Section 2 gives the reader an understanding of the starting point of this research project and what alterations have been made to the existing setup.

Section 3 presents an overview of the system setup, describing both the hardware and software that makes up the emulator and the components meant to interface with it. It also gives a basic introduction to wireless communication and the limitations introduced by each of the components.

Sections 4 to 6 provides the reader with an understanding of what has been done to improve the existing emulator and sensor nodes from the work

by Danielsson and Ågren. Section 4 describes the modification of the user interface, including adding features to import and export data. In section 5, a solution to the emulator’s latency issue is presented where the Media Access layer (MAC) and Radio Duty Cycling (RDC) protocols of the sensor nodes are altered. Section 6 describes an experiment conducted to show how interference and noise models can be implemented using the existing hardware. In summary, by programming a sensor node to emit a noise floor and connecting the sensor node to the emulator as normal, it is possible to mimic many different sources of noise and interference.

Conclusions from the project are listed in section 7 and section 7.4 identifies open issues.

2 Related work and contributions

This section describes how this project relates to some similar projects and outlines how it contributes to a previous project on physical layer emulation.

2.1 Related work

A proof-of-concept version of an emulator for the physical layer of the wireless communication of sensor networks was developed by Danielsson and Ågren at Uppsala university [2]. They used off-the-shelf hardware and the Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW) software to create an emulator that supports real time, two-way communication between up to eight IEEE 802.15.4 [3] compliant sensor nodes. The results of the previous work was encouraging, but the following issues remained:

- The emulator introduced a latency to any signal fed through it. In practice, this latency caused sensor nodes to time out while waiting for acknowledgments of received transmissions which was interpreted as a transmission failure.
- The simple user interface limited what tasks could be performed with the emulator.
- There was only minimal support for on the fly loading of measured channel data. If the emulator is to be used as intended, a simple way to load channel data is required.
- There was no means to save the results of an emulation session for later review. Such a feature is essential in an analysis of the behavior of the emulator throughout a test.
- There was no means to apply external interference. Having the ability to apply an external interference to the recorded channel data would make the emulator capable of performing more advanced tests.

2.2 Contributions

The purpose of this project was to address the issues presented by Danielsson and Ågren [2] and outlined in section 2.1. The following goals were achieved:

- Input-output latency issues: By altering the MAC and RDC protocols of the sensor nodes, the latency issues of the emulator have been remedied. Previously, the input-output latency of the emulator caused sensor nodes to time out while waiting for a reply. By resolving this issue, the analysis and evaluation of more advanced communication schemes has been made possible. These communications schemes are a requirement for many applications, as they provide the means for retransmitting data packets that do not reach the receiver. In this report, the steps performed in altering

the protocols are described and an attempt is made to enlighten the reader about the necessary steps and precautions when doing so.

- User interface: To increase the usability of the emulator, the software has been modified to be easier to use. All controls and indicators have been clearly labeled and their specific functions have been documented. Through discussions and test runs with end-users, the usability of the new user interface has been verified.
- Importing channel data: A data format for storing measured channel and site data has been proposed. It allows storage of relevant data, and can be adapted to suit future needs. In conjunction with the newly designed data format, a simple way to import previously measured channel data from file and apply these channels in the emulator has also been created. As with the data format, the import tool can easily be modified to suit a changing need.
- Saving signal data: It is now possible to save the transmitted and received data of every sensor node connected to the emulator. The data is saved continuously while running the emulator and stored in a comma separated list.
- Interference and noise sources: It has been shown that it is possible to implement different interference models in hardware by connecting an interference source to one of the emulators inputs. It has also been demonstrated that it is possible to use a sensor node as the source of noise or interference.

3 System overview

This section describes both the hardware and software components that make up the emulator, as well as any components meant to interface with it. The hardware used throughout this work is identical to the hardware used by Danielsson and Ågren in [2]. Also included in this section is a basic introduction to wireless communication and the limitations introduced by each component.

The system consists of up to eight IEEE 802.15.4 compliant sensor nodes, a pair of power splitters/combiners, and the emulator hardware, all of which is monitored and controlled from a PC running a custom made software. The sensor nodes can, optionally, be connected to the PC via a Universal Asynchronous Receiver/Transmitter (UART) connection, allowing the monitoring of sensor node output. The PC is connected to the emulator via an ExpressCard interface to the Peripheral Component Interconnect Express (PCIe) bus.

The emulator itself has three major components. A Radio Frequency Signal Analyzer (RFSA) captures and samples all transmissions from the connected sensor nodes. The sampled transmissions are processed by a Field-Programmable Gate Array (FPGA), and finally output by a Radio Frequency Signal Generator (RFSG), allowing the sensor nodes to receive the transmissions.

A description of the setup is provided here. The data flow of the system set up is illustrated in figure 1.

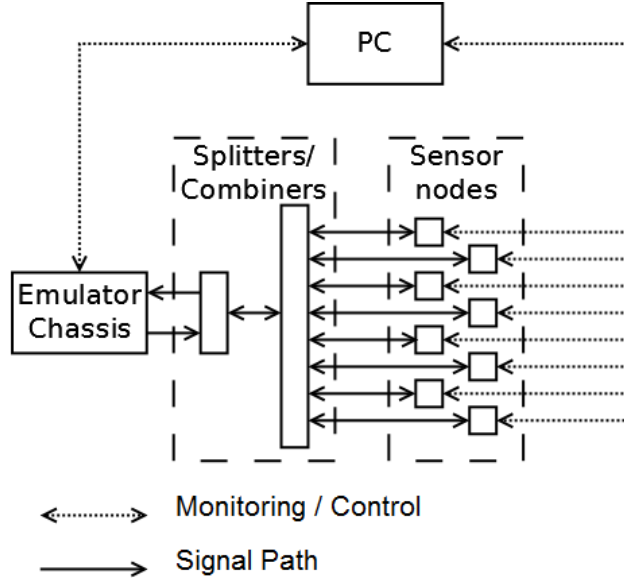


Figure 1: Signal and data monitoring flow of the system setup. Via separate interfaces, the PC sets up the behavior of the emulator as well as any sensor nodes connected to it. Both the emulator and the individual nodes are able to relay control information back to the PC. Both the input signals to, and the output signals from the emulator are combined into a single signal by the combiners. The individual signals are extracted by tuning a receiver to the carrier frequency and bandwidth of the desired signal.

3.1 Sensor nodes

The sensor nodes used throughout the work described in this report has been the Zolertia Z1[4], depicted in figure 2.

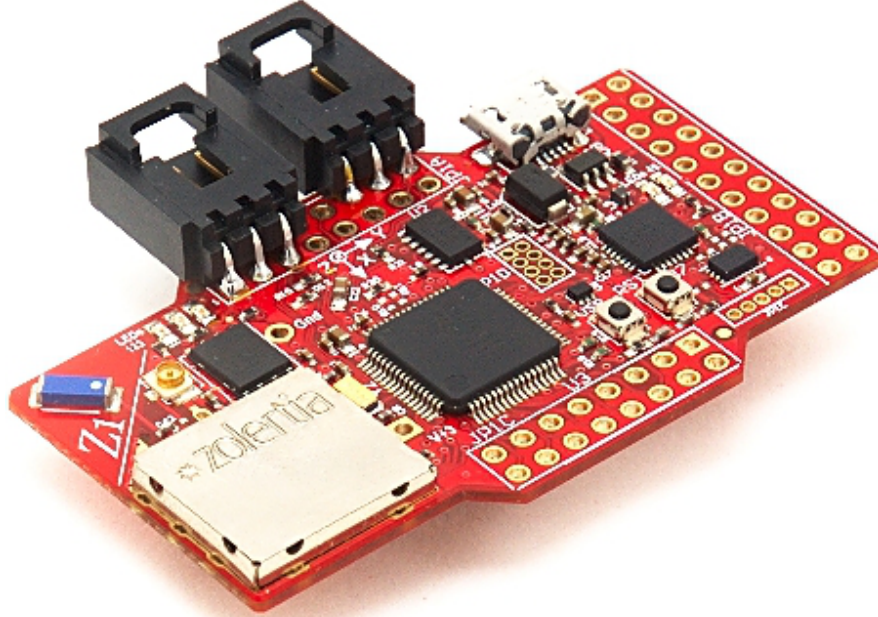


Figure 2: The Zolertia Z1 sensor node uses the IEEE 802.15.4 compliant radio transceiver CC2420.

There is no reason to believe that the same results should not be achievable using any IEEE 802.15.4 compliant sensor node, as long as precautions are taken to adapt for any differences in hardware. The Zolertia Z1 is equipped with the IEEE 802.15.4 compliant radio transceiver CC2420, giving the sensor node its means of communication. The IEEE 802.15.4 standard defines the properties of the physical layer and the media access control (MAC) of low-rate wireless personal area networks and, as such, it is the principal standard for a number of energy efficient wireless systems, including the Zolertia Z1. A brief summary of the CC2420 radio transceiver is provided in table 1. For more detailed information, refer to the complete CC2420 data sheet [5].

Table 1: CC2420 radio transceiver specifications.

RF frequency band	2400.0–2483.5 MHz
Transmit bit rate	250 kbps
Output power	−25–0 dBm
Receiver sensitivity	−95 dBm
Saturation level	10 dBm

The CC2420 radio transceiver can communicate either via an embedded

ceramic antenna, or via an external antenna attached via a U.FL connector [6]. Which antenna is used is determined by the placing of an output capacitor on the circuit board of the Zolertia Z1, see figure 3.

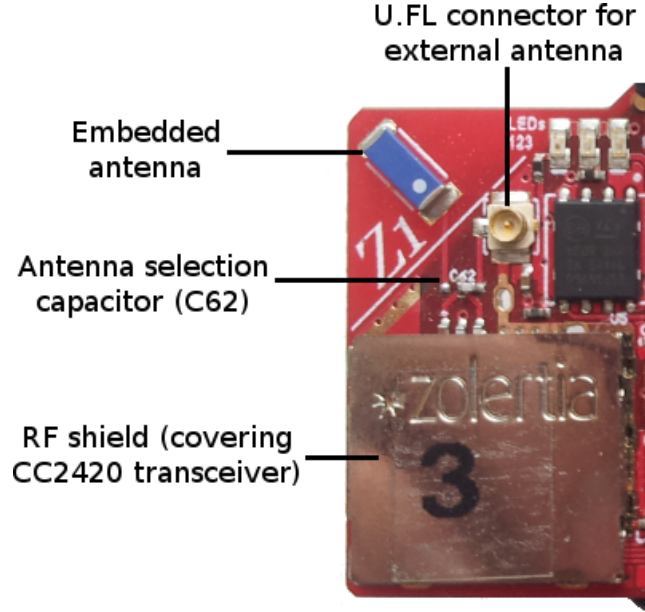


Figure 3: A close up view of the Zolertia Z1’s RF components. Note especially the capacitor marked as C62, here soldered to provide the output signal on an external antenna via the U.FL connector.

A brief summary of the specifications of the Zolertia Z1 sensor node is provided in table 2. For more details refer to the IEEE 802.15.4 standard [3] or to the Zolertia Z1 data sheet [4].

Table 2: Zolertia Z1 specifications.

MCU	MSP430F2617
Transceiver	CC2420
Antenna	Embedded or external via U.FL connector
Built-in sensors	3-Axis accelerometer, Temperature sensor
Power source	Battery or USB connector

3.1.1 Over-the-air transmissions

In order to verify that the emulator is producing reasonable results, it is important to understand how the sensor nodes work during normal over-the-air transmission and to determine a way to measure the performance and quality of a wireless channel. To estimate the quality of wireless channels the Package Reception Ratio (PRR) has been used for both over-the-air transmissions and transmissions made through the emulator. PRR indicates the proportion of successfully received packages over time. Assuming any bit errors are uncorrelated and the probability of bit error p_{ber} , or Bit Error Rate (BER), is known, PRR is calculated as

$$PRR = (1 - p_{ber})^n \quad (1)$$

where n is the number of bits transmitted. The packages transmitted during these tests were all of the size $n = 22 \text{ bytes/package} \cdot 8 \text{ bits/byte} = 176 \text{ bits/package}$, including both header and payload bits. In IEEE 802.15.4, see annex E of [3], the probability of bit error is a function of the Signal to Interference and Noise Ration ($SINR$) of the received signal and is given by

$$p_{ber} = \frac{8}{15} \cdot \frac{1}{16} \cdot \sum_{k=2}^{16} (-1)^k \binom{16}{k} e^{(20 \cdot SINR \cdot (\frac{1}{k} - 1))}. \quad (2)$$

$SINR$ is defined as the ratio between the power of the received signal, P_r , and the sum of the interference, I , and noise, N , powers, i.e.

$$SINR = \frac{P_r}{I + N}. \quad (3)$$

Both noise and interference power levels are highly dependent on the environment of the receiver. A too low $SINR$ will result in a higher rate of failed transmissions and would, at a certain point, make the sensor network unusable. Maintaining a high $SINR$ on all channels is therefore a prime objective when designing a sensor network.

Under standard operations the sensor nodes would be transmitting over the air. The signal strength of these transmissions is inversely proportional to the distance between the two antennae and can be modeled coarsely by a simplified path loss model. Let P_t and P_r denote the transmitted and received powers at the respective antenna, and d the distance between the two antennae. The received power is then given as

$$P_r = P_t K \left(\frac{d_0}{d} \right)^\gamma, \quad (4)$$

where d_0 is a reference distance, and γ is the path-loss exponent. The range of γ is usually between 2, for free space transmission, and 6, in areas where transmissions are heavily obstructed by walls or other objects. When using the CC2420 radio transceiver, P_t can be user configured to values in the range

$[-25 \text{ dBm}, 0 \text{ dBm}]$. The combined antenna gain of both antennae, K , is a function of the wavelength, λ , and the reference distance, d_0 ,

$$K = \left(\frac{\lambda}{4\pi d_0} \right)^2. \quad (5)$$

It is clear that the received signal strength will decrease as the distance and path-loss exponent (i.e. the amount of obstruction between the transmitter and receiver) increases. As the received power decreases, so does the receivers *SINR* and *PRR*.

This theoretical model is used in section 6.2 to verify the behavior of the system when a noise source is introduced.

3.2 Peripheral equipment

In order to connect the sensor nodes to the input and output of the emulator, a set of coaxial antenna cables and a pair of power splitters/combiners are needed. The antenna cables are connected to the U.FL connector of the sensor nodes, which therefore must have its output capacitor soldered to connect an external antenna. The two splitters/combiners are connected back-to-back, see figure 4, in order to fully interconnect the eight sensor nodes, as well as the input and output of the emulator.



Figure 4: The back-to-back connection of the power splitters/combiners allows the simultaneous connection of up to eight sensor nodes to both the input and output connectors of the emulator.

The splitters/combiners are both the same type of component, but with opposite orientation. When used as a combiner, they are designed to take the superposition of signals from all ports on one side and place the combined signal on the single port on the opposite side. Correspondingly, when used as a splitter, the input signal on the single port is output simultaneously on all output ports. It is important that these components are designed to work well, i.e. minimal distortion and damping, within the 2.40–2.48 GHz frequency range used by the emulator.

The power splitters/combiners are designed to provide a 0° phase shift of the signals passed through it. They also introduce an insertion loss proportional to the amount of inputs of the specific splitter/combiner. This would mean that a combination of a 2-to-1 and an 8-to-1 splitter/combiner would give a theoretical insertion loss of

$$10 \cdot \log_{10}(2) + 10 \cdot \log_{10}(8) = 12.0 \text{ dB.} \quad (6)$$

Measuring the total insertion loss of a signal in the mentioned frequency range, passed through the splitter/combiner setup has yielded results of about 12.7 dB.

Please refer to [7] and [8] for further information on the specifications of the splitters/combiners used.

When connected by these components the sensor nodes are essentially directly connected to each other with a minimum loss of power in received transmissions, caused by the transmitted signals double pass through the splitter/combiner setup (one pass from sensor nodes to the emulator, and one pass from the emulator back to the sensor nodes), resulting in a base loss of signal strength of about 25.4 dB.

3.3 Emulator

The emulator is constructed using commercially available hardware from NI. All of the hardware used is part of the NI PXIe series of hardware, meaning that the modules are able to communicate using the PCIe communication buses. The signal travels through the emulator as illustrated in figure 5.

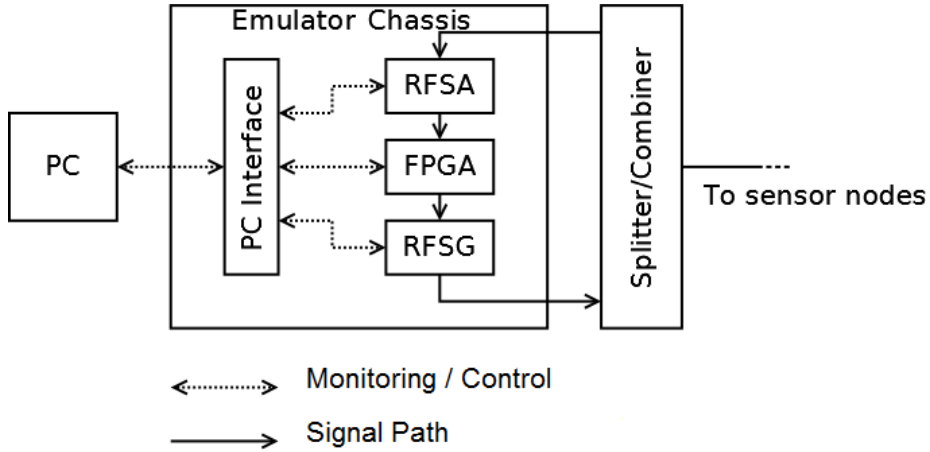


Figure 5: The internal signal and data monitoring flows of the emulator. The transmissions are converted and sampled by the signal analyzer (RFSA), after which the sampled signal is processed by the FPGA before being output again by the signal generator (RFSG).

The blocks labeled RFSA and RFSG refer to the vector signal analyzer and vector signal generator, respectively and are described further in section 3.3.1.

3.3.1 Emulator Components

As indicated in figure 5, the emulator consists of a signal analyzer for sampling the sensor nodes output signals, an FPGA for applying the channel gain matrix, a signal generator for outputting the processed signals, as well as some other components described further in the following paragraphs.

NI PXIe-1078 Chassis The NI PXIe-1078 Chassis, described in detail in [9], provides a compact basis for connecting the remaining NI hardware components together and provides the PCIe links to the attached modules. The maximum theoretical data rate of any module connected through the PCIe link is 200 MB/s.

NI PXIe-5663E Vector Signal Analyzer The NI PXIe-5663E Vector Signal Analyzer, described in detail in [10], constitutes the input part of the emulator. The sum of all analog signals from any connected sensor node is down mixed and then filtered through a band-pass filter in order to avoid any aliasing in the analog to digital converter (ADC). The sampled signal is then passed through the PCIe link to the NI PXIe-7966R FlexRIO FPGA module where the channel gain matrix can be applied.

NI PXIe-7966R FlexRIO FPGA An FPGA is a type of integrated circuit that can be configured by the user after manufacturing. This ability offers great flexibility when compared to more traditional application-specific integrated circuits (ASIC). The drawbacks of implementing your circuits as an FPGA rather than an ASIC are, generally, increased physical size (which, in turn, affects cost levels), decreased performance and increased power consumption. In [11], Kuon and Rose present data comparing these metrics for FPGAs and ASICs of equal transistor size. They find that an FPGA that uses only lookup tables and flip-flops is, on average, 40 times bigger, 3.2 times slower and uses 12 times more power than an ASIC. The differences decrease somewhat if specialized hardware, such as multipliers, adders and dedicated memory blocks that would be needed by practically any conceivable implementation, is used in the FPGA. Despite the increased cost and decreased performance compared to ASICs, FPGAs can still be a very useful and viable alternative, especially for prototyping or other cases where you might want to alter the configuration at a later time.

The NI PXIe-7966R FlexRIO FPGA module, described in detail in [12], uses a Virtex 5 FPGA device from Xilinx [13].

The FPGA module applies the appropriate, time varying, channel gains to each input signal and provides the information that is to be displayed in the LabVIEW control software. The input signal generated by the ADC in the NI PXIe-5663E Vector Signal Analyzer is down mixed and separated from its composite form to eight separate signals,

$$X(k) = \begin{pmatrix} x_1(k) \\ \vdots \\ x_8(k) \end{pmatrix}. \quad (7)$$

The eight signals are then multiplied by an eight-by-eight channel gain matrix,

$$C(k) = \begin{pmatrix} c_{11}(k) & \cdots & c_{18}(k) \\ \vdots & \ddots & \vdots \\ c_{81}(k) & \cdots & c_{88}(k) \end{pmatrix}, \quad (8)$$

containing the channel gains that are to be applied between each pair of sensor nodes. As a sensor node is unable to transmit to itself the elements of the main diagonal are all zero, i.e. $c_{ii}(k) = 0, i = 1, \dots, 8, \forall k$. Also, since the channel between sensor node i and sensor node j is identical to the channel between sensor node j and sensor node i , the matrix is symmetric, i.e. $c_{ij}(k) = c_{ji}(k)$. Knowing this we can simplify $C(k)$ as

$$C(k) = \begin{pmatrix} 0 & c_{21}(k) & \cdots & c_{81}(k) \\ c_{21}(k) & \ddots & \ddots & \\ \vdots & \ddots & \ddots & c_{87}(k) \\ c_{81}(k) & \cdots & c_{87}(k) & 0 \end{pmatrix}. \quad (9)$$

The output signals, $y_1(k), \dots, y_8(k)$, that result from the matrix multiplication are linear combinations of the eight input signals, as determined by the channel gain matrix,

$$Y(k) = \begin{pmatrix} y_1(k) \\ \vdots \\ y_8(k) \end{pmatrix} = C(k) X(k). \quad (10)$$

These resulting signals are finally upconverted, superimposed into a single signal and passed to the NI PXIe-5673E Vector Signal Generator. In order to present data to, and enable packet monitoring in the user interface, the FPGA module also calculates the absolute values of the input and output signals and makes this information available to the LabVIEW host user interface.

NI PXIe-5673E Vector Signal Generator The NI PXIe-5673E Vector Signal Generator, described in detail in [14], provides the output part of the emulator. The digital signal received from the NI PXIe-7966R FlexRIO FPGA module is passed through a pair of digital to analog converters (DACs). The signal is reconstructed by feeding it through a low-pass filter and up mixing it to a center frequency of 2.425 GHz.

NI PXIe-8360 Laptop Control module The NI PXIe-8360 Laptop Control module, described in detail in [15], provides a completely transparent link between the emulator hardware and the control software running on a PC. The PC and the chassis communicate through a PCIe link with a maximum sustained throughput of 214 MB/s. Note that all signal processing is performed within the emulator and the PC only provides the controls for properly starting the emulator and setting the channel gain matrix, as well as presenting the signal and channel data to the user.

3.3.2 Transmitting through the emulator

Using the emulator, it is possible to emulate the physical layer of the wireless communications between up to eight connected sensor nodes. There are, how-

ever, a couple of issues in relation to the frequency channels used by the sensor nodes.

Under normal, over-the-air, transmissions all sensor nodes would, typically, be configured to transmit and receive on a single frequency channel. However, as there is some latency in the emulator, a sensor node would be able to switch from transmitting to receiving mode in time to hear its own transmission. Since all signals, both output and input, ultimately travel through the same antenna cable, there is also the risk of a feedback loop being created between the emulators input and output ports. Some sort of shielding would also be required to prevent sensor nodes from receiving transmissions before they pass through the emulator. To solve this issue, the sensor nodes have been programmed so that each node transmits and receives on two different channels, i.e. every sensor node uses two wireless channels for communicating with the network. This means that before and after transmitting, a sensor node must change its frequency channel.

In addition to this, there is also the issue of how the emulator can identify a given transmission as originating from a certain sensor node, and output that signal with different channel gains to different sensor nodes. The issue was solved by assigning a connected sensor node its own, unique, pair of frequency channels for transmission and reception, leading to a total of 16 frequency channels, each with a bandwidth of 5 MHz, being used by the emulator to mimic the behavior of over-the-air transmissions. This use of the available frequency spectrum has led to further changes in how the sensor nodes must behave in order to conform with the emulator's expectations. See table 3 for a clarification on how frequencies are arranged.

Table 3: The assignment of carrier frequencies for each of the eight sensor nodes attached to the emulator. Each frequency channel is 5 MHz in bandwidth. Each of these channels are represented by a number specified by the IEEE 802.15.4 standard.

Sensor node	Receive frequency	Channel Number	Transmit frequency	Channel Number
#1	2402.5 MHz	11	2442.5 MHz	19
#2	2407.5 MHz	12	2447.5 MHz	20
#3	2412.5 MHz	13	2452.5 MHz	21
#4	2417.5 MHz	14	2457.5 MHz	22
#5	2422.5 MHz	15	2462.5 MHz	23
#6	2427.5 MHz	16	2467.5 MHz	24
#7	2432.5 MHz	17	2472.5 MHz	25
#8	2437.5 MHz	18	2477.5 MHz	26

Another issue that relates to the latency between input and output of the emulator is the way the sensor nodes interpret the delay they experience. The delay will only cause problems when the sensor nodes of the network are using communication protocols that implement acknowledgments of received trans-

mission. In this case, a delay of tens of micro seconds could be interpreted as the distance between two sensor nodes being several kilometers. Under normal conditions, the signal would be far too weak to travel such distances and with that in mind, the sensor nodes communications protocols have not been designed to handle such delays. This has been one of the main problems to solve in order to make the emulator more usable, and it is discussed further in section 5 together with a proposed solution.

3.4 Software tools

A number of software tools have been used during the course of this work, both for development and simulation. The possible applications of these tools are extensive and not limited to what is described here. This chapter is meant to be a summary of the work done using these tools.

3.4.1 LabVIEW

LabVIEW is NI's development environment for the G programming language. G is a data flow programming language and, as such, it describes how things connect. This can be compared to the way imperative programming languages like, for example, C, C++ or Java, describe how things happen. Because of the different design of data flow programming languages, certain tasks are much more easily accomplished than with imperative programming languages.

An application created with LabVIEW is referred to as a virtual instrument (VI). LabVIEW also promotes designing your application in a modular way by making a previously developed VI a part of another application. This is done in a black box manner, where inputs and outputs are made available from the contained VI, usually referred to as a subVI, to the containing VI, and the internal workings of the subVI are mostly obscured to anyone running the application. The creation of the user interface is integrated in the development of the VI, this will be described in more detail in section 4.

LabVIEW also contains several toolboxes for integrating with many different types of instrumentation hardware, like signal generators, data acquisition hardware and FPGAs from both NI and other vendors.

3.4.2 Contiki

Contiki was first created by Adam Dunkels at the Swedish Institute of Computer Science, and is an open source operating system for hardware constrained, low power systems using wireless communication. Contiki fully supports standard IPv4 and IPv6 networking through the uIP and uIPv6 network stacks allowing communication over the internet. In addition to this, Contiki also supports a number of additional protocols and standards for low power wireless communication, making it ideal for use with devices and for applications with limited power supply.

Contiki supports a range of different hardware platforms from a number of manufacturers, including the Zolertia Z1 sensor node used throughout this work and described in further detail in section 3.1.

The Instant Contiki development system is distributed in the form of a virtual machine and provides all the compiler tool chains and libraries required to develop a system running Contiki, as well as a number of tools useful for debugging large networks of wireless devices, such as the simulation software Cooja.

3.4.3 Cooja

Cooja is a simulation environment for wireless sensor networks. It offers the ability to perform large scale tests using a variety of sensor nodes, either by simulating the nodes or through full hardware emulation. Within Cooja, information on the state of the sensor nodes in the simulation is made easily available. This information can then be accessed and used for the purpose of testing and debugging.

Using a simulation tool when developing and designing a sensor network is invaluable for early testing. It needs to be pointed out, however, that while much information can be gained through the simulation of sensor networks, a simulator can never be a complete model of every aspect of the environment. The flaws of simulated environments constitute the reason behind the project presented in this report.

4 Modifying the LabVIEW host interface

A host interface is the name given to the user interface of a virtual instrument (VI) created in LabVIEW. It presents acquired data, and offers control parameters to the user, which enables the user to manipulate the operation of the VI.

Part of the specification of this work involved reworking the user interface of the previously available LabVIEW host interface. This involves both cosmetic changes to the user interface, like rearranging, relabeling or redesigning the available user interface, as well as introducing new functionalities through modifications to the existing VI and subVIs, or through the addition of completely new subVIs.

The functionality of added features have been verified as an ongoing process throughout the development of these features. The full documentation and user instructions for these features are available in Appendix B.

The redesign of the user interface has undergone several iterations, each time collecting feedback on the different features. The feedback has been incorporated in the current design of the user interface.

This section describes the alterations made to the existing user interface. They include both visual changes and added features. Section 4.1 describes the feature used to import previously measured channel data from file, as well as the file format designed for use with this feature. Section 4.2 discusses the feature of signal data export and describes the format of the output file. Section 4.3 gives an overview of the new layout created for the user interface. Section 4.4 outlines how the verification of the new user interface was done.

4.1 Importing channel data

The greatest alteration to the user interface has been the addition of the ability to easily import channel data from file. Using pre-measured data for repeatable testing is one of the main reasons for developing the emulator.

After measuring the wireless channels at a certain site, channel gain data is generally processed in Matlab and stored in a Matlab data file. Previous to the work done here, no definitive format for these files had been determined, meaning it would be impossible for a completely automated system to reliably process these files and evaluate the data therein. Therefore, the first task of implementing the data import function is establishing a standard format for the files used to store the channel gain data to be imported to the emulator.

The information that may need to be stored in the data file is firstly general data and information about the site and collective data on the sensor nodes, and secondly specific information about each measured channel. Also, as the number of wireless channels measured at different sites can vary greatly, the data must be in a format that allows it to be expanded to any size. The data format suggested here can easily be modified and the corresponding changes to the LabVIEW files can also be implemented quickly, if needed. The current implementation is illustrated in table 4.

Table 4: The channel data file structure. The initial fields are for site and test information, followed by an array of structure arrays, each element containing the information of a specific channel.

fs	
siteInfo	
	receivingNode transmittingNode rssi[] channelInfo
	⋮
	receivingNode transmittingNode rssi[] channelInfo

The data pertaining to every channel is contained within an array, **s**, of Matlab structure arrays and each channel can be uniquely identified by the combination of transmitter and receiver node IDs (**transmittingNode** and **receivingNode**, respectively). A time series of captured Received Signal Strength Indication (RSSI) values is stored in the **rssi[]** array. This array can be of arbitrary size. Finally, the **channelInfo** field gives the ability to store any arbitrary information that may be relevant to the measured data, such as a verbal description of the channel's propagation path, or a string that allows you to easily identify a specific channel. Outside the array of structure arrays information relevant to the entire test site and the entire system of sensor nodes can be stored, specifically, **fs** is a floating point value representing the sampling frequency in Hz used during the data collection, **siteInfo** is a string containing description and miscellaneous information about the test site where the data was collected. The site, and all channel information data is kept within an outer structure array, **p**. This helps organize the data by gathering it in a single data object.

The application that parses the data file is implemented as a subVI inside the main LabVIEW application. During the parsing, the data is expected to be structured as described above and the subVI will keep parsing the data file for channel data until it reaches the end of file. At that point the subVI will exit and make any data extracted available to the containing VI. The main application will list all channels found in the data file and index them from 0 to $N - 1$, where N is the number of channels successfully parsed from the data file. Any channel parsed from the data file can then be applied between any pair of sensor nodes by specifying the appropriate index in the channel control matrix in the main application.

More information on the channel data import subVI is available in appendix B.

4.2 Exporting sensor signal data

Another major feature that has been added is the ability to save sensor node input and output to file. Signal data will be written continuously to a file specified by the user. This file will be in the form of a comma separated list, with sixteen values for each row. Each row represents a point in time and each value represents the signal strength detected on this channel at this time.

When writing to an empty file, the string values “sensorOut0”, ..., “sensorOut7” and “sensorIn0”, ..., “sensorIn7” will be written, in sequence, to the first row of the file. This is to simplify the import of the data file into Matlab or any similar piece of software for data processing.

The VI handling the output of sensor signal data could easily be expanded to log more data as new needs are discovered.

More information on the channel data import subVI is available in appendix C.

4.3 General modifications

For increased overview of the available controls and indicators, the different parts of the host interface have been grouped by functionality into separate tabs. This grouping provides a more intuitive way of finding relevant controls and data, while at the same time decreasing the general clutter of the host interface. In addition to the grouping, all available components of the host interface have been relabeled in a more consistent and descriptive way, along with their functionality being documented in appendix A.

4.3.1 Signal monitoring

The signal monitoring tab contains simultaneous plots of all input or output signals, along with controls for selecting the trigger channel and number of samples to be plotted. Also part of this tab are the newly added options for logging signal data to file.

The plots show the average of the signals absolute value over a number of samples. This operation is performed by the FPGA. Using the notation of equations (7)-(10), we can express each point in time of the plotted sensor output and input signal values as

$$\langle |X(k_n)|, \dots, |X(k_m)| \rangle, \quad (11)$$

and

$$\langle |Y(k_n)|, \dots, |Y(k_m)| \rangle \quad (12)$$

respectively. Where $(k_i)_{i=n}^m$ is a series of time points and the indices m and n represent the bounds of the time period over which the average is taken. In [2], the interval is chosen so that $m - n = 64$. During the work on this project it was found that this interval produces excessive amounts of data, especially when attempting to log signal data to file. Ideally, increasing this interval would be done by modifying the code that is being run on the FPGA.

However, modifying this code was outside the scope of this project. All signals are therefore downsampled by a factor 16 after being extracted from the FPGA, and before being presented on screen or written to file. It was found that even after this downsampling, it was still possible to visually detect packages at least as small as 11 bytes, and possibly even shorter.

Using the Trigger length control it is possible to determine how many such values are plotted at any given time.

4.3.2 Channel monitoring

The channel monitoring tab gives the controls to select the behavior of the channels, as well as a plot to monitor the absolute values of the channel gains. In particular, when using the new function for importing channel data, you now have the ability to select which of the previously measured time series is to be applied to each individual channel.

4.3.3 Channel information

The channel information tab displays any information that may have been parsed from a data file, including both information about site and channels. This is particularly useful when determining which channels should be applied between which two sensor nodes. The entire concept for this tab was implemented in this version of the emulator host interface.

4.3.4 Status

The status tab attempts to present the status information of the hardware, especially information pertaining to the signal generator and signal analyzers states. This information is used mainly to monitor for over- or underflow issues, something that may be important to catch during further development of the emulator host interface but should not be of any concern to the typical end-user.

4.3.5 Hardware setup

The hardware setup tab is used to configure the behavior of the hardware components, and to specify the correct devices to be used. It is very important that the correct devices are selected and appropriate values configured.

4.3.6 Errors

Under the errors tab any errors encountered during the running of the emulator are presented. A number of new error types were introduced during this thesis project. To facilitate debugging the host interface is able to catch and distinguish between nine different types of errors.

4.4 Verification of the host interfaces usability

Through bench testing, and through feedback from potential end-users, the function and usability of the host interface in its present form has been verified.

4.4.1 Importing channel data

The subVI that handles importing channel data from file is described in detail in Appendix B. The functionality of the subVI was continuously tested and verified during the development of the feature. As long as the data file being processed is of the format described in section 4.1, data sets of any size may be extracted. The subVI processes the data until it reaches the end of file or an error occurs. The time series stored in the data file need not be of equal length, as each time series is parsed and presented individually.

Data files of an unexpected format will typically cause errors as the subVI attempts to interpret the available data. It is, however, possible for the subVI to handle a data file which contains more fields than those proposed in section 4.1, provided the additional data fields are all stored after the expected data fields. Any additional data parsed in this way will simply be discarded. Designing the subVI in this way enables more flexibility of the file format used without any modification necessary to the subVI.

The final version of the subVI was tested by converting a number of available, previously gathered, data sets containing the type of channel information that would be used when performing tests with the emulator. Results showed no issues with importing channel data.

4.4.2 Exporting signal data

The subVI that handles the output of signal data to file was continuously tested during development. This testing highlighted the problem with excessive amounts of data outlined in paragraph 4.3.1 and was the main reason for downsampling the signal before plotting and exporting the signal data.

It was also verified that the exported signal data could be imported into Matlab. This gives the possibility for later review of signal data produced during experiments with the emulator.

4.4.3 General modifications

Objectively verifying the increased usability of the redesigned host interface is difficult without feedback from end-users. These end-users should also, ideally, be familiar with both the previous and current versions of the host interface. Because of the difficulties in achieving this large scale testing, the improvements to the interface have instead been measured by comparing the current state of the host interface with the expectations of the potential future users of this system.

Through this feedback features such as context sensitive user inputs have been added, and labeling and scales of the various controls and indicators have been improved.

5 Latency

This section discusses the problem with acknowledgement transmissions associated with the emulators input-output latency and proposes a solution to this issue. Section 5.1 describes how the latency problem was verified, and section 5.2 describes how it was addressed.

It was mentioned in [2] that the latency of the emulator may introduce problems with certain types of transmissions. The total latency for a single transmission through the emulator was measured to be $42.18\ \mu\text{s}$. During the tests performed in [2], simple protocols for radio duty cycling (RDC) and media access layer (MAC) were used that kept the transceiver powered at all times and had no means available to detect failed transmissions. It was suspected that, when using more sophisticated communication protocols, this latency will cause the sensor nodes to time out while waiting for other nodes to acknowledge their transmission, thus assuming transmission failure and initiating retransmission before the acknowledgement has had enough time to reach the intended receiver.

5.1 Verifying the latency

The default implementation of the ContikiMAC protocol, as specified by [16], leaves room for latencies of up to $48\ \mu\text{s}$ for protocols that implements acknowledgments of transmissions. Given the previously measured latency of $42.18\ \mu\text{s}$ for a signal that passes once through the emulator, it is likely that the nodes will experience problems with communication while connected through the emulator and employing any communications protocol that relies on acknowledgments of received transmissions, i.e. protocols that require more than one pass through the emulator and thus experience a latency of at least $84.36\ \mu\text{s}$.

To verify that the latency is an issue that needs to be addressed, tests must be performed using some standard set of protocols that implements acknowledgement of received transmissions. However, further issues arise immediately as the implementation of the channel emulator requires transmission and reception to happen on different frequency channels. Because of this, the transmitting node must switch channel while waiting to receive acknowledgement from the receiver, and the receiver must switch channel when transmitting the acknowledgement. This is something that is not supported in the standard protocols, as transmissions and reception over-the-air are physically required to happen on the same channel. The available RDC and MAC protocols must therefore be modified to support this sort of behavior.

Here the CX-MAC protocol has been used as a starting point for creating a protocol that will function with the emulator. The main reason for choosing CX-MAC over ContikiMAC is that the former handles acknowledgments in software, while the latter handles acknowledgments in hardware. Handling acknowledgments in hardware is faster but modifying the behavior of the protocol to the needs of the emulator would require more work.

After modifying the CX-MAC protocol to add channel switching, an experiment was performed where a total of 1000 transmissions were sent at a rate of

5 transmissions per second. During the experiment the PRR was used to measure the performance of the setup. PRR is defined as the number of correctly received data packets divided by the total number of sent data packets. The experiment showed a PRR of 0.7 % which is a clear indication that the latency caused by the emulator is a significant problem that must be addressed.

5.2 Adding latency tolerance to the system

It was suggested in [2], that upgrading the emulator with the NI PXIe-5644R Vector Signal Transceiver module, which integrates an FPGA, RFSG and RFSA into a single module, could remove most of the latency caused during the internal I/O of the emulator. The details of this module can be found in [17]. The latency caused by internal I/O in the emulator has been measured to be $19.55 \mu\text{s}$, meaning that the latency could potentially be reduced by as much as 46 %. However, because of the financial investment that would be involved in exchanging the hardware used, this option has not been explored further.

The issue was resolved, instead, by altering the MAC and RDC protocols used by the sensor nodes. The solution involved modifying the timings of the protocol in order to allow the transmissions to be completed successfully even with the delay caused by the emulators latency.

5.2.1 Altering the MAC protocol

In addition to adding the channel switching to the RDC protocol, the timings of the protocol must also be altered in order to compensate for the latency caused by the emulator. A guard time of $70 \mu\text{s}$ was added before a sensor starts listening for the acknowledgement of successful transmission. This time was selected based on a series of tests wherein guard times of different lengths were used in an attempt to find an optimum. A total of 1000 transmissions were made for each guard time. The plot in figure 6 shows the PRR achieved using guard times between 0.00-1.15 ms.

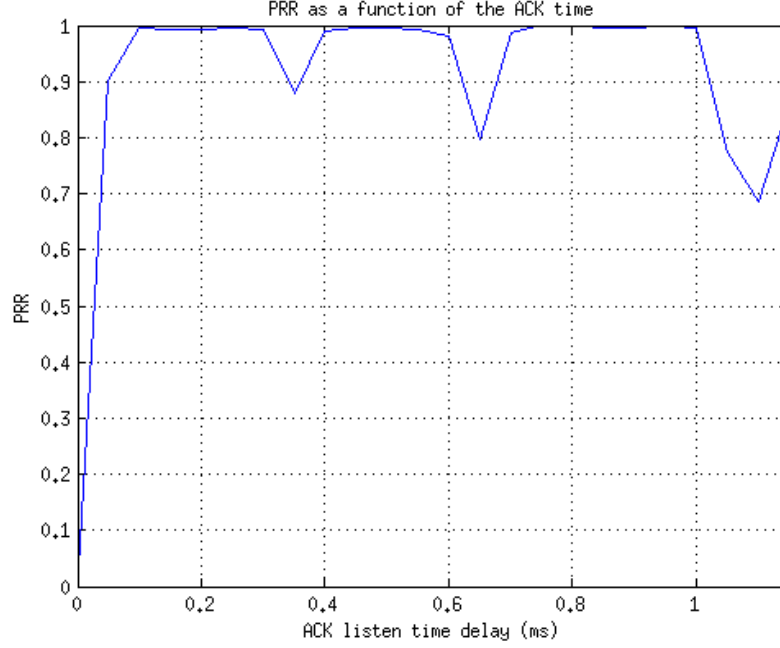


Figure 6: The resulting PRR based on different waiting times for acknowledgement reception. Note the periodical decrease in PRR that occurs every 0.3-0.4 ms

The PRR appears to follow a periodic behavior depending on the increase in acknowledgement wait time. The precise reason for this behavior is likely related to the changes in the timing of the sensor nodes interactions and should be possible to anticipate and avoid with further knowledge of the CX-MAC protocol.

Before each data transmission performed by a node A, it will attempt to establish contact with the recipient node B. Node A will make several attempts to transmit a short preamble message, each time going into receiving mode for a predetermined amount of time to listen for an acknowledgement of successful transmission from node B. Once node B has successfully received a the preamble from node A, node B will send an acknowledgement of the received transmission addressed to node A. Each such acknowledgement is followed by node B waiting to receive the data transmission that is to follow. If node A successfully receives the acknowledgement from node B, node A stops transmitting preambles and transmits a data package. In the CX-MAC implementation, no acknowledgement is sent from node B after it receives the data transmission. The behavior of a pair of nodes, A and B, is illustrated in figure 7.

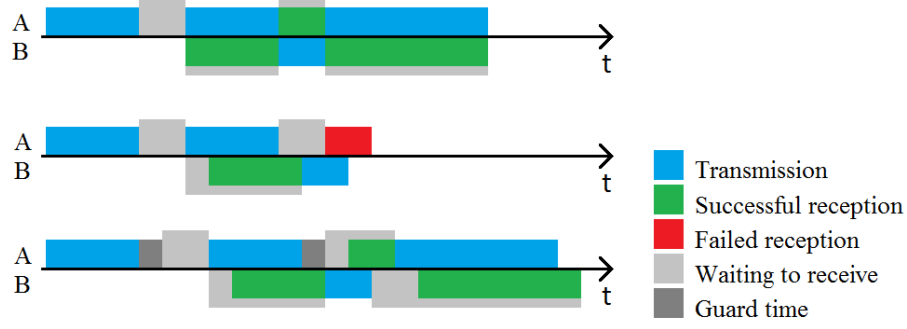


Figure 7: The behavior of two nodes, A and B, as they attempt to complete a packet transmission under different conditions. The upper case is over-the-air transmission, the middle case is transmission through the emulator without the modified protocols, and the bottom case is through the emulator while using the modified protocols. Note that relative sizes of the different blocks are exaggerated to illustrate the nodes transmission behavior and do not accurately reflect the relative times it takes to perform the operations.

In the upper case, node B wakes up and is instantly able to receive the preamble from node A, and after replying with an acknowledgement node B also receives the data transmission. In the middle case, the emulators latency first causes a delay before node B is able to receive the preamble, the acknowledgement sent from node B also experiences the same latency and node A times out before receiving the acknowledgement. Finally, in the bottom case, the preamble and acknowledgement are still affected by the latency of the emulator, but the guard time introduced at node A ensures that node A does not start listening for the acknowledgement unnecessarily early.

A second test with higher resolution on the time interval 0.05-0.14 ms was also performed to further narrow down the optimum guard time of the acknowledgement listening. The results of this second test is plotted in figure 8.

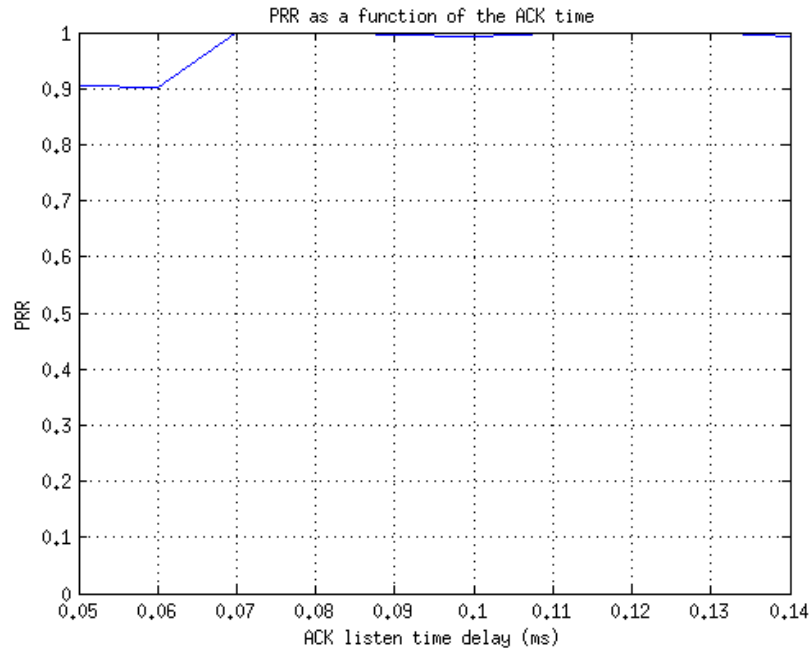


Figure 8: The results of the second measurement of PRR, using a higher resolution scale compared to the first test.

Based on these results a guard time of $70\mu s$ was chosen to minimize the overall latency of the system.

6 Adding interference/noise sources

This section describes a way to introduce noise and interference sources to the emulator without modifying the existing signal processing done by the emulator. Section 6.1 describes an experiment conducted in relatively noise free environments. Section 6.2 describes a similar experiment where a sensor node is configured to output a noise floor of increasing power.

By programming a sensor node as an interferer or noise source, i.e., to output data that disturbs the transmissions of other sensor nodes, the base tests can be expanded with the addition of different type of disturbances. This is helpful when attempting to design more robust sensor networks that need to maintain performance under changing conditions. This approach has been investigated previously, for example by Boano et al. in [18].

Other types of hardware, with greater power output and better spectrum control, could also be interesting as sources of interference.

6.1 Establishing a baseline PRR

To measure the effects of transmitting with an interference source, a baseline is first established by measuring the PRR for transmissions under different conditions. Baseline PRRs were measured for simulated, over-the-air, and through the emulator transmissions without any interference source. The simulations were performed using Cooja, described in section 3.4.3. All three tests were performed using the CX-MAC protocol in simulation and over-the-air experiments and the modified CX-MAC protocol described in section 5.2.1, in the case of the emulator, with approximately 10000 transmissions and a transmission rate of 1 Hz. During the simulation and over-the-air experiments, sensor nodes were placed as close together as possible, approximately 1 cm for over-the-air. When using the emulator, the close placement of the sensor nodes is emulated by setting the channel gain to 1. The PRRs achieved in these measurements are listed in table 5.

Environment	PRR
Simulation	100.0 %
Over-the-air	98.4 %
Through the emulator	98.3 %

Table 5: PRRs achieved during transmission tests where the only interference source is the background interference present.

The perfect PRR of the simulation environment is to be expected for the idealized, and interference free conditions that exists in the simulator. For the over-the-air transmissions, the sensor nodes embedded antennae were used. The antennae were aligned with each other and kept at a distance of about 2 cm. Exactly what the dropped packets in the air and emulator transmissions are

caused by is uncertain, but it is probable that some of the packet losses are caused by external interference or thermal noise in the RF circuits. The similar results achieved with both over-the-air and through the emulator transmissions indicates that the emulator closely resembles the conditions of over-the-air transmissions.

6.2 Connecting a sensor node as a noise source

By placing the DACs of the CC2420 transceiver in a test mode, it is possible to program a sensor node to emit a constant noise floor, see [5] for details. A single sensor node was programmed to perform this way and output noise floors of increasing levels. At each level 1000 packets were transmitted between two different sensor nodes through the emulator. These transmissions were performed with a transmission power $P_t = -12$ dBm, sending one packet every second, and using the modified CX-MAC protocol. The results of these measurements are listed in figure 9 along with the theoretical PRR, calculated as outlined in section 3.1.1.

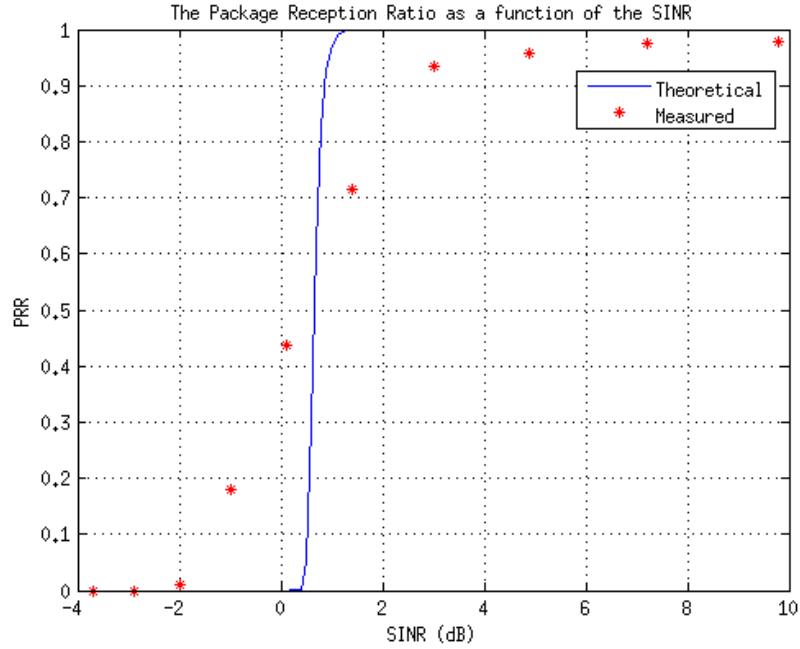


Figure 9: A comparison between theoretical and measured PRR. The two curves are similar, although the theoretical values seem to have a much sharper rise as SINR increases.

It is clear that the increased noise floor contributes negatively to the PRR.

There is a decrease in the measured PRR at an SINR of 3 dB and at an SINR of -2 dB all transmissions are effectively blocked.

By following the work outlined in [18], sophisticated interference sources can be created to imitate the interference patterns of other devices that may be present in the 2.4 GHz frequency band. A simpler way to achieve the same results may be to use the new feature of channel data importing to create a similar behavior. Manually create a series of channel data representing the behavior of an interference source and configure a sensor node to output a constant noise floor, and applying this channel data to a link between the interfering sensor node and a sensor node operating normally, the noise floor created by the interfering node is modulated by the artificial channel data and will appear as an interferer of the desired type. It is possible to create any number of interference patterns for use in this way, using only a single sensor node as the source of the interference signals. This method also removes the need to reconfigure the sensor node before being able to apply a new interference pattern. The limiting factor in this setup is the speed at which the emulator can update the channel data. Currently, the emulator is hardwired to fully update the channel matrix every 10 ms. As a consequence, if a rapidly changing interference pattern is desired, it must still be implemented as a hardware solution.

7 Summary and conclusions

The contributions, as outlined by section 2.2, closely follow what has been the desired outcome of this project.

7.1 Host emulator interface

Evaluating the changes to the interface of the host emulator is a difficult task that requires extensive user testing. For various reasons, performing these tests has not been a part of this work, instead potential end-users have been allowed to provide feedback on the user interface and features as the work has progressed.

7.1.1 Channel data import

The new subVI created for importing data has been verified as working with data files that conform to the data format suggested in section 4.1. Since the final specifications for this data format is yet to be determined, the current design of the subVI has been implemented with a minimal amount of additional features. For example, no safeguards have been implemented to ensure predictable behavior when attempting to parse data files that do not follow the expected data format. Instead, the aim has been to implement a minimal working subVI that can be further refined once the final specifications of the project have been clearly established.

7.1.2 Export of resulting data

As with the channel data import subVI, the signal data export subVI has been verified to be working and outputs data files of the format outlined in section 4.2. Adding support to this subVI for the export of additional data should be trivial, if the need to do so arises.

7.2 Latency

After investigating different ways of addressing the latency issue, the problem was finally solved by altering the CX-MAC protocols radio duty cycle to allow for a longer waiting time for transmission acknowledgments. Along with the increased wait time, the protocol was also altered to make data reception and acknowledgement transmissions happen on different channels. This was necessary due to the design of the emulator, where emulator input and output happens on different channels.

Tests performed with the altered protocol have shown a substantial increase in PRR, to a point where transmissions only fail very occasionally.

If there would be a need to actually reduce the latency, not just mitigating it, the option of upgrading the hardware is still available. Using a module such as the NI PXIe-5644R, which integrates the FPGA, RFSA and RFSG into a single device could potentially remove nearly half of the existing latency. Also, as is mentioned in section 5.1, the protocol developed in this work handles

acknowledgments in software and not in hardware. It is generally true that lower level implementations are faster than their higher level counterparts. It is reasonable to assume that it would be possible to reduce the latency by finding a way of implementing the necessary channel switching within a protocol that handles acknowledgments in hardware.

7.3 Interference sources

It has been shown that a sensor node can be successfully configured to behave as an interference source. By configuring the DAC's of the sensor nodes transceiver to a test mode, the sensor node will output white noise. Implementing added interference in this way is a quick and simple way to introduce this useful feature without any modifications to the emulator.

The basic noise floor can be expanded and modified to imitate those of WiFi, Bluetooth, microwave ovens, and more simply by creating artificial channel data to modulate the basic noise floor output of the sensor node.

7.4 Further work

The emulator, in its current state, still leaves room for improvement and further work. Some of the aspects that might be considered for improvement are outlined below.

7.4.1 Further testing and development

Although some tests have been performed during the course of this work, it is very likely that more will be required before the emulator can be seen as complete. A design specification should be established that clearly specifies the desired features of the finished product. In order to accomplish this, an end-user needs to be able to identify the real world applications where this piece of equipment might be useful, so that features might be added to support these applications.

In addition to any further work on the emulator, more work can be done on refining the MAC and RDC protocols used by sensor nodes. For example, moving the acknowledgement implementation, and the channel switching associated with it, from software to hardware would decrease system latency. This might require exchanging the currently used RDC protocol for the ContikiMAC protocol. Furthermore, developing software for the sensor nodes that can be used for extensive testing of the sensor nodes performance in certain network constellations and with certain channel data applied would be very beneficial. Such software would greatly increase the usefulness of the emulator by allowing a quick and easy way to run the sort of extensive tests that this equipment was initially designed to run.

It would also be interesting to work further on the disturbance sources. This could be done either by creating series of channel data matching the behavior of certain interference sources, and later verifying these as reliable sources of

interference, or by creating or acquiring new hardware specifically for use as disturbance sources.

References

- [1] Glenn Judd and Peter Steenkiste. Using emulation to understand and improve wireless networks and applications, 2005.
- [2] Anton Danielsson and Jakob Ågren. Utveckling av generisk kommunikationsplattform för trådlösa sensornätverk. Master's thesis, Uppsala University, 2013.
- [3] IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), 2009.
- [4] Zolertia. Z1 Datasheet. <http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf>, March 2010.
- [5] Texas Instruments. 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. http://www.crew-project.eu/sites/default/files/CC2420_Data_Sheet_1_4.pdf, April 2006.
- [6] Hirose Electric Group. Ultra Small Surface Mount Coaxial Connectors. http://www.hirose.co.jp/cataloge_hp/e32119372.pdf.
- [7] Mini-Circuits. Power Splitter/Combiner ZAPD-4+. <http://www.minicircuits.com/pdfs/ZAPD-4+.pdf>.
- [8] Mini-Circuits. Power Splitter/Combiner ZB8PD-4+/ZB8PD-4. <http://www.minicircuits.com/pdfs/ZB8PD-4.pdf>.
- [9] National Instruments. NI PXIe-1078 9-Slot 3U PXI Express Chassis. <http://sine.ni.com/ds/app/doc/p/id/ds-312/lang/en>, 2013.
- [10] National Instruments. Vector Signal Analyzer. http://www.ni.com/pdf/products/us/cat_PXIe_5663.pdf.
- [11] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. In *FPGA '06 Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 21–30, 2006.
- [12] National Instruments. NI FlexRIO FPGA Modules. <http://sine.ni.com/ds/app/doc/p/id/ds-366/lang/en>.
- [13] Xilinx. Virtex-5 Family Overview. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, Febuary 2009.
- [14] National Instruments. Vector Signal Generator. http://www.ni.com/pdf/products/us/cat_PXIe_5673.pdf.

- [15] National Instruments. Laptop Control of PXI (ExpressCard MXI for PXI Express). http://www.ni.com/pdf/products/us/cat_pxie8360.pdf.
- [16] Adam Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical report, Swedish Institute of Computer Science, December 2011.
- [17] National Instruments. NI PXIe-5644R/5645R 6 GHz RF Vector Signal Transceivers. <http://sine.ni.com/ds/app/doc/p/id/ds-422/lang/sv>.
- [18] Carl Alberto Boano, Thiemo Voigt, Claro Noda, Kay Römer, and Marco Zúñiga. Jamlab: Augmenting sensor network testbeds with realistic and controlled interference generation. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks*, April 2011.

A Emulator description

This appendix is meant to serve as a guide for any end-user wanting to use the emulator in its present form.

A.1 User manual

A.1.1 Hardware setup

To ensure that the hardware is connected as required, please follow these steps:

1. Plug in the ExpressCard controller interface in the appropriate slot of your PC, see figure 10.



Figure 10: The ExpressCard controller interface connects the emulator and your PC.

2. Make sure the cable running between the emulator and your PC is secured at both ends.
3. Connect the 2/1 and 8/1 power combiners/splitters back-to-back, see figure 11.



Figure 11: Make sure to connect the power splitters/combiners as pictured.

4. Connect the two free ports of the 2/1 power splitter/combiner to the output and input ports of the emulator, see figure 12. Either of the ports of the power splitter/combiner may be connected to either of the ports on the emulator.

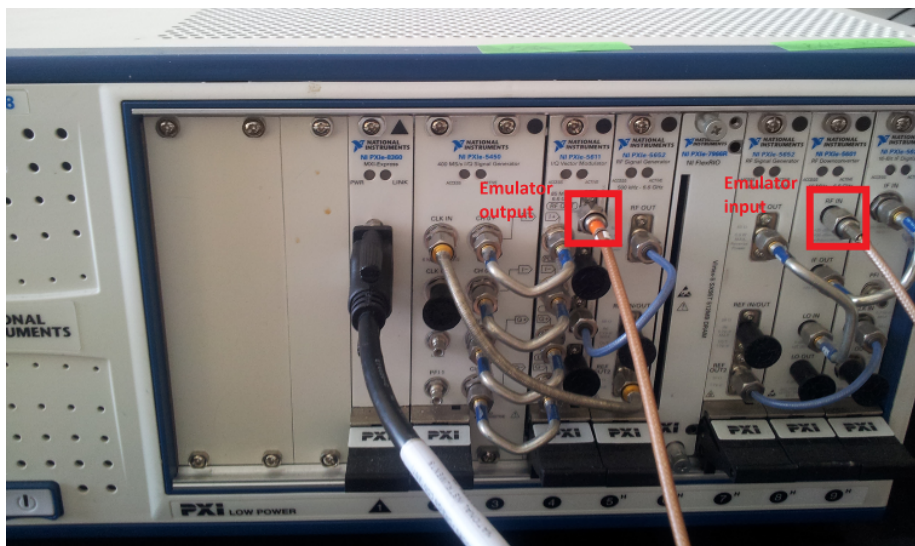


Figure 12: Connect the power splitter/combiner to the emulators input and output ports.

5. Connect up to eight sensor nodes to the free ports of the 8/1 power combiner/splitter using a U.FL compatible connector, see figure 13.

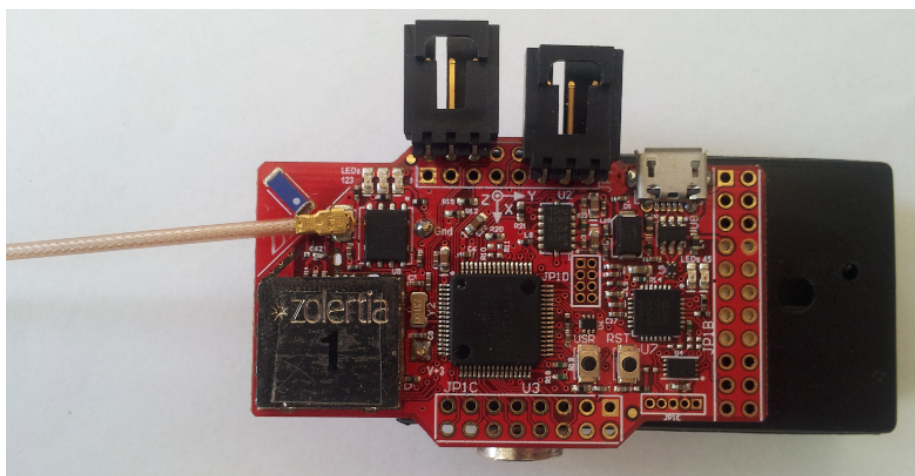


Figure 13: Each sensor node must be connected to a port on the 8/1 power combiner/splitter.

6. Optionally, you may connect any number of your sensor nodes to your PC over USB. This will give you the ability to capture any serial output from

the sensor node(s).

7. Power on the emulator chassis.
8. Restart the PC.

A.1.2 Required software

Make sure you have working copies of the following National Instruments softwares:

- LabVIEW 12.0
- NI-DAQmx
- PXI platform services
- LabVIEW FPGA module
- LabVIEW Real-time module
- NI-RIO
- NI-RFSA

This is most easily done by using the Measurement and Automation Explorer from Nation Instruments, see figure 14.

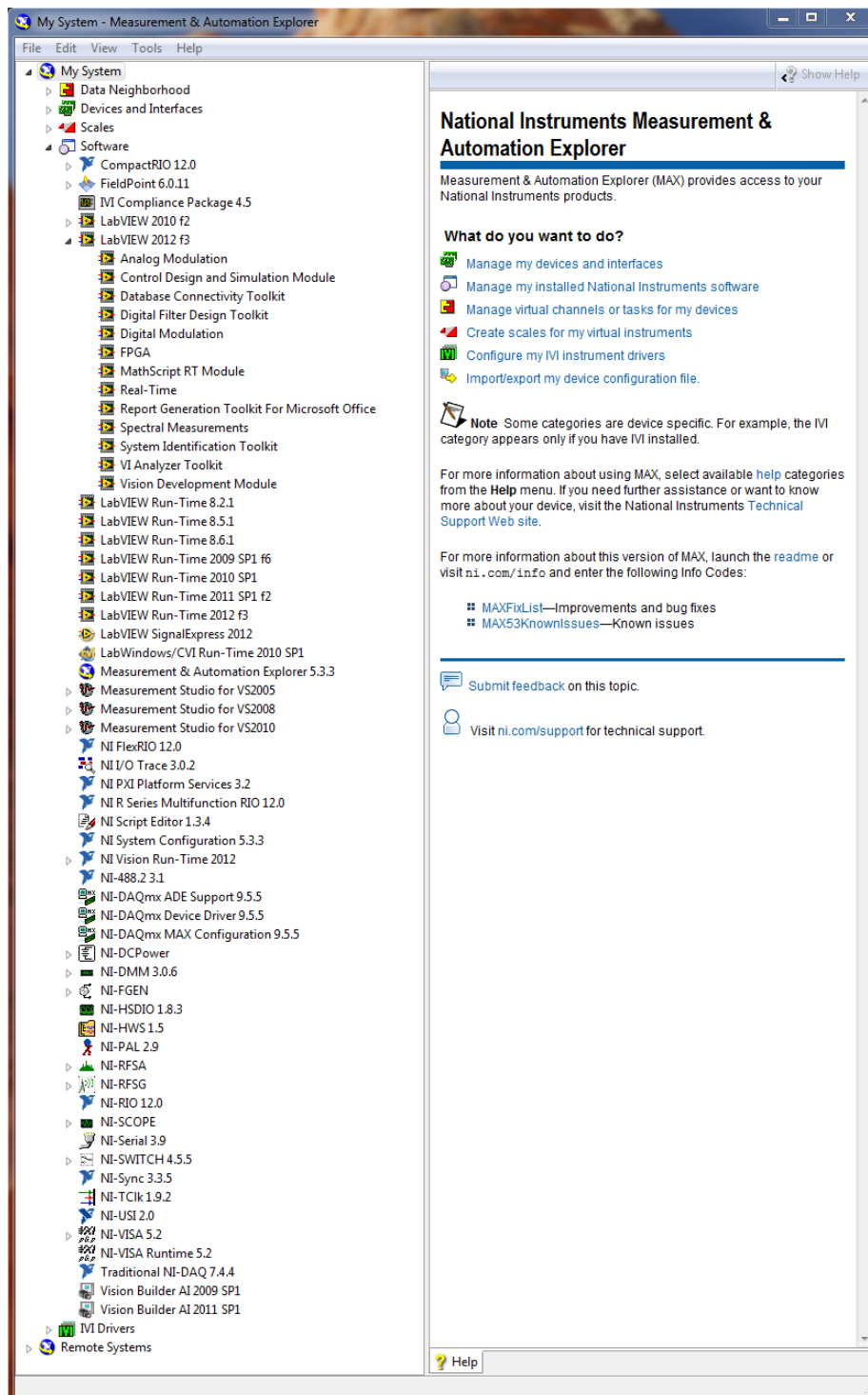


Figure 14: Use the Measurement and Automation Explorer to find installed software from National Instruments.

Also, ensure that you have a complete version of the emulator host interface. The complete version should include the following files:

- addOffset.vi
- channelSelectorReciprocity.vi
- channelSort.vi
- config_instruments.vi
- emulatorHost.vi
- generateChannelData.vi
- importData.vi
- saveSignalData.vi

as well as the FPGA bitfile `ChannelSimulator_PXIe-7966R_CHMatrix(8Ch).lvbitx`.

The VI-files should all be placed in a single folder and will be imported into your LabVIEW library when you run the emulator.

Finally, open the emulator host interface by loading the file `emulatorHost.vi` in LabVIEW.

A.1.3 Configuring the emulator

Before running the emulator host interface, make sure you configure it to suit your needs.

- Go to the **Hardware Setup** tab of the host interface.
- Make sure the settings for NI-FPGA Device, NI-RFSG Device, and NI-RFSA Device are correct for your setup. You may need to refer to the **Devices and Interfaces** part of the **Measurement & Automation Explorer**, see figure 15.

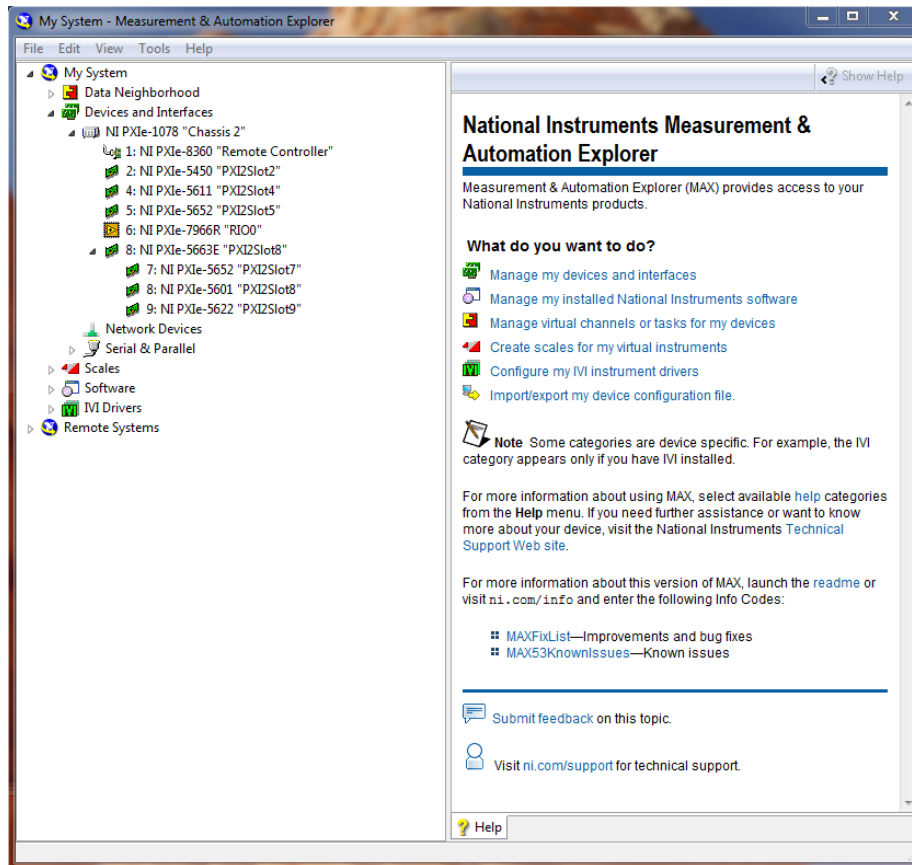


Figure 15: In the Devices and Interfaces part of the Measurement & Automation Explorer you should be able to identify the hardware components of the emulator.

- The remaining settings available on the Hardware Setup tab should be configured as shown in figure 16.

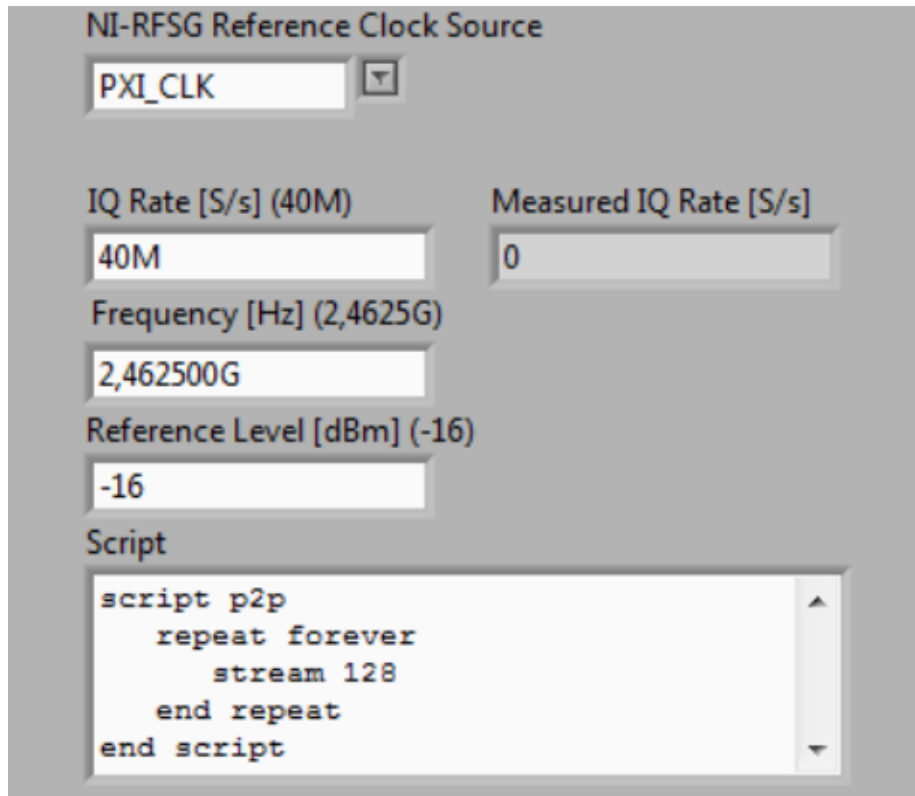





Figure 16: Default values of the remaining hardware setup configuration options.

- Next, go to the Channel Monitoring tab of the host interface.
- Select the Channel mode you wish to use.
- Except for the Read data from MATLAB file option, all options are self configuring.
- If you choose to use the Read data from MATLAB file option, you need to follow these additional steps:
 - Select a valid, previously generated, data file using the available Data file field
 - If you are unsure of the ordering of your channel data within the selected file, you can briefly run the host interface by clicking the run button () , wait a few seconds and then click the stop button () . This will generate a list of all available channel data under

the Channel Information tab. You can refer to this list during the next step.

- In the Channel Selector matrix, specify the index of the channel data you wish to associate with a particular pair of sensor nodes. Note: the indices should always be entered in the upper triangle of the Channel Selector matrix, as the emulator will later mirror this half of the matrix to create a symmetric system of channels.
 - Finally, use the Number of Samples to read data field to specify how much of the available channel data is to be read from the data file. Note: once the final sample has been applied, the channel gain will remain constant at this value for the remainder of the execution.
 - For users wanting to make several runs with the same set of channel data applied may want to use an existing LabVIEW feature to make the entered values the default values. Simply right click the control you wish to give a new default value and select Data Operations»Make Current Value Default, see figure 17.
- Next, go to the Signal Monitoring tab of the host interface.
 - Use the Sensor Output/Sensor Input toggle control to select whether you would like to monitor the connected sensor nodes output or input signals. Note: this can be changed during runtime.
 - If desired, configure the Trigger channel and Trigger level to suit your needs and turn triggering on.
 - To enable saving of signal data to file, toggle the Save signal data to file? input and specify an output file. Note: it is possible to append new data to an already existing file, provided the data of the existing file is of the same format as the new data to be written to the file.
 - You are now ready to start the emulator host interface by pressing the run button ().

A.1.4 Initializing the sensor nodes

Depending on the setup and configuration of your experiment, you may need to finalize the setup of the sensor nodes by doing the following steps.

- If you wish to monitor the serial output of any sensor node, make sure the node(s) are connected to your PC via USB and open up a connection to the sensor nodes serial interface. Note: make sure to use the correct baud rate when connecting to the sensor nodes (typically 115200).
- You may wish to restart the sensor nodes when initializing your experiment. Nodes typically have a reset button integrated on their circuit board, see figure 18 for an example.

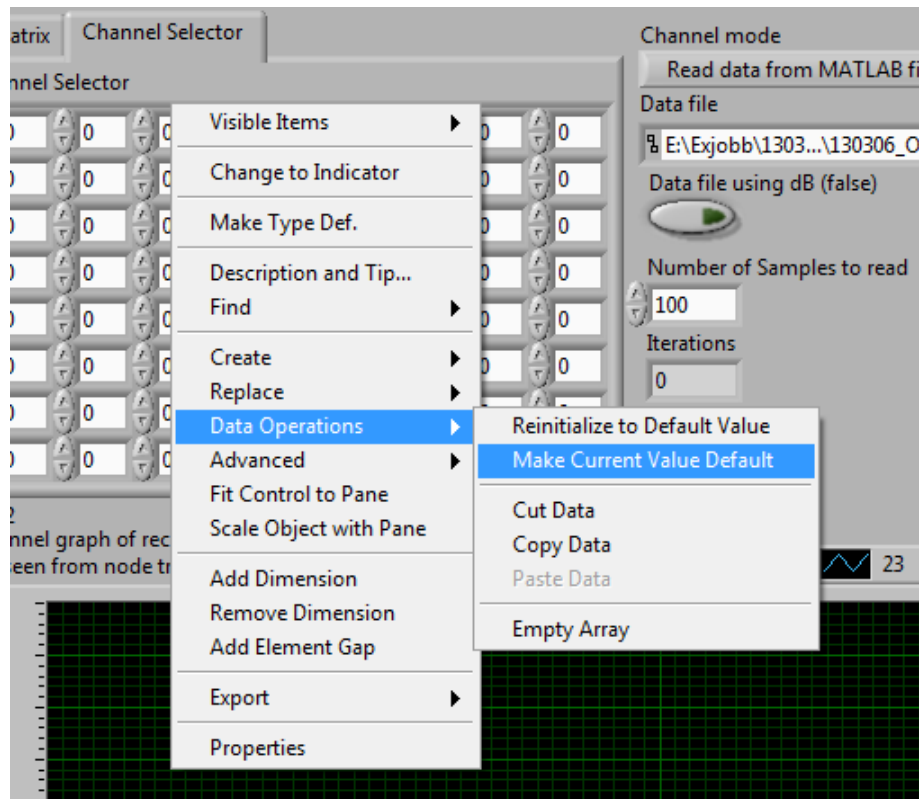


Figure 17: Use the right click dialog to change the default values of your controls.

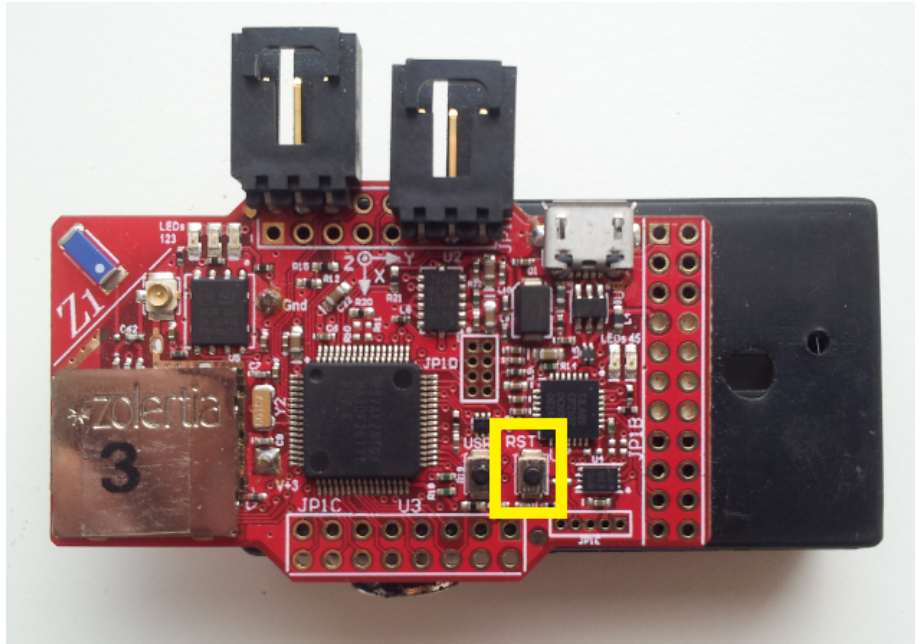



Figure 18: The reset button of the Zolertia Z1 sensor node. Sensor node appearance and circuit board layout may differ between manufacturers.

A.1.5 Stopping execution

To stop the execution of the emulator, simply press the stop button (.

A.2 Emulator Host Controls and Indicators

These are the functions of the controls and indicators of the channel emulators host interface, grouped by tab.

A.2.1 Signal Monitoring

This tab gathers controls and indicators for monitoring the emulators input and output channels. Pictured in figure 19.

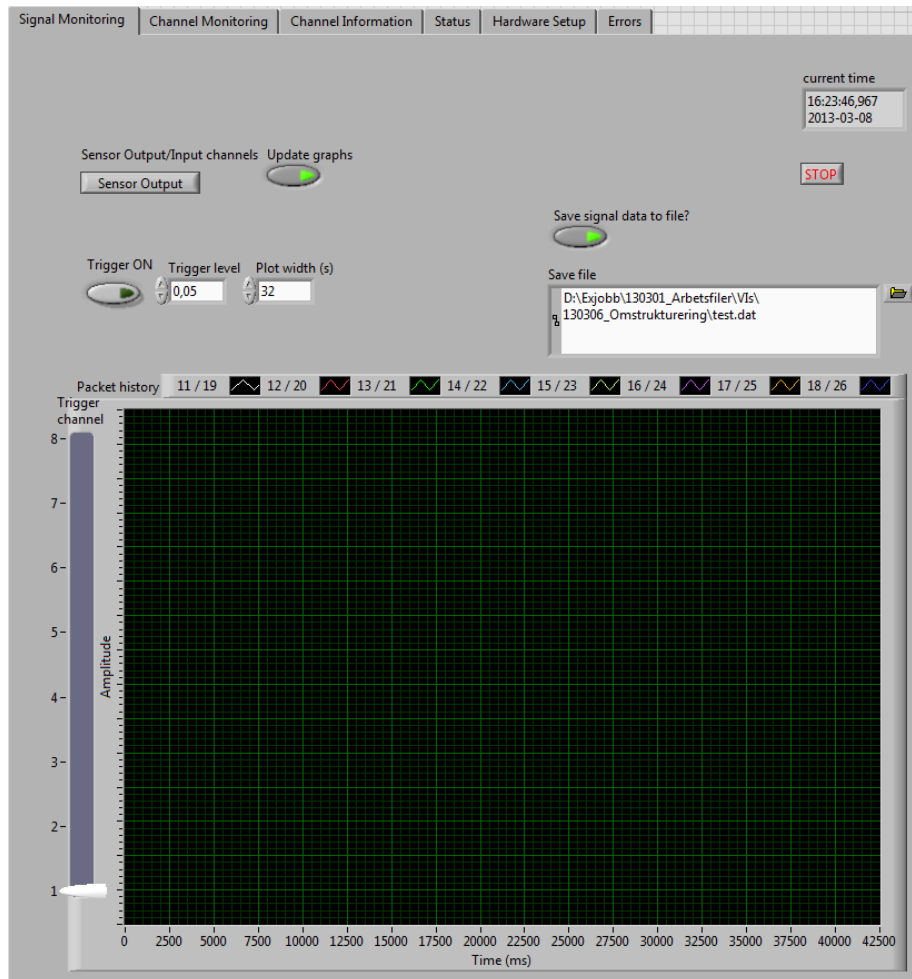


Figure 19: The signal monitoring tab of the emulator.

- Input/Output channels
 - Selects whether the emulators input or output channels should be plotted.
- Update graphs
 - Selects whether the plots should be updated or not. Only pauses the plots, not the emulator itself.
- Trigger ON
 - Selects whether the trigger function should be used or not.

- Trigger level
 - The smallest amplitude required to initialize the trigger function.
- Trigger length
 - How many samples that are to be plotted. The number of plotted samples are 1024 times the trigger length.
- Packet history
 - Plots all eight of the emulators input or output signals. The amplitude of all plots has an offset added to it, in order to make the plot easier to read.
- Trigger channel
 - Selects the channel to trigger on.

A.2.2 Channel Monitoring

This tab gathers controls and indicators for monitoring and setting the values of the emulated channels. Pictured in figure 20.

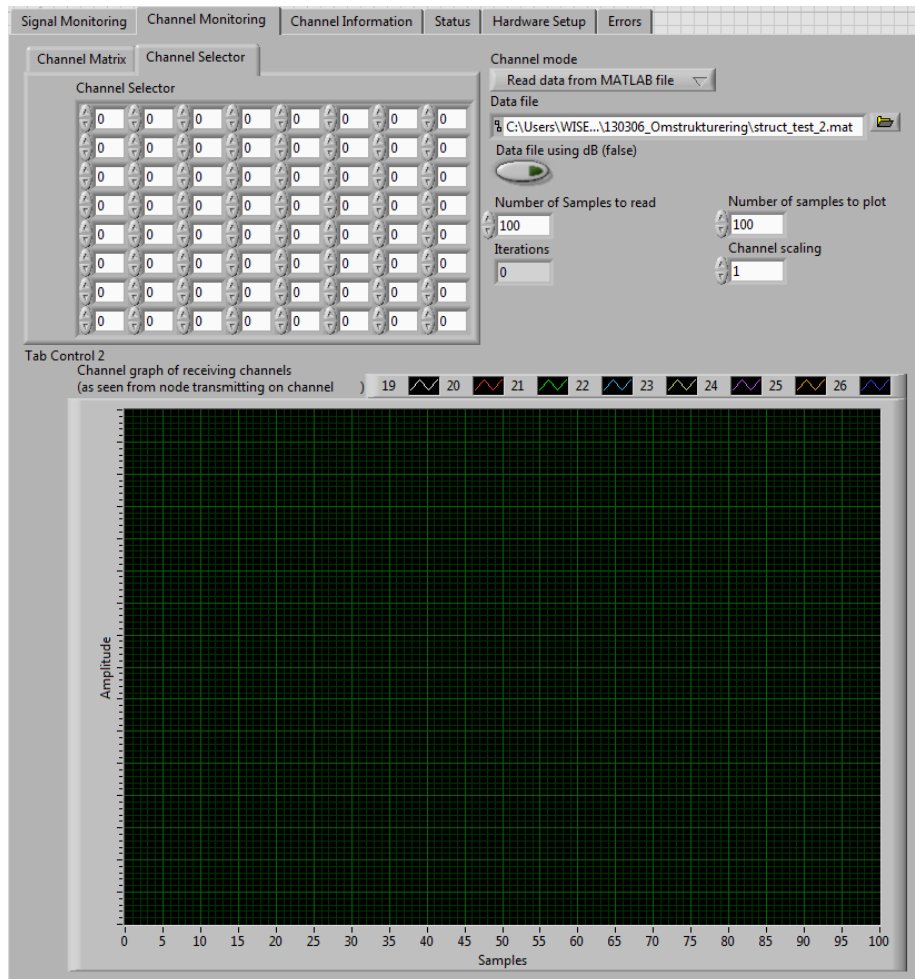


Figure 20: The channel monitoring tab of the emulator.

- Channel matrix
 - Shows the current values of all the channel coefficients.
- Plotted channels
 - The Row of channel coefficients that are to be plotted.
- Channel selector
 - Specifies which channel, read from data file, is to be assigned to each channel.

- Channel mode
 - Selects the mode by which the channel coefficients should be set. Current available modes are:
 - * Read data from MATLAB file
 - * Diagonal
 - * All constant
 - * Random channels
- Data file
 - The data file to read channel data from.
- Data file using dB
 - Not implemented.
- Number of samples to read
 - The number of samples to read from the data file.
- Number of samples to plot
 - The number of samples to be displayed in the plot.
- Iterations
 - Number of times the channel matrix has been updated. For debugging purposes.
- Channel scaling
 - Can be used to scale down the amplitude of all coefficients of the channel matrix.
- Channel graph
 - Plots a time series of eight channel coefficients, as experienced by signals transmitted from one specified sensor node.

A.2.3 Channel Information

This tab contains information on the channel coefficient data parsed from a data file. Pictured in figure 21.

The screenshot shows a software interface with a top navigation bar containing tabs: "Signal Monitoring", "Channel Monitoring", "Channel Information" (which is selected), "Status", "Hardware Setup", and "Errors". Below the navigation bar, the "Channel Information" section is divided into two main areas. The top area, labeled "Site cluster", contains a "Sampling frequency [Hz]" input field with the value "0" and a "Site info" label above a large, empty rectangular box. The bottom area, labeled "Channel cluster", contains a table with four columns: "Channel index", "Receiving node", "Transmitting node", and "Channel info". Each of the first three columns has four input fields, all containing the value "0". The "Channel info" column has four empty rectangular boxes. Below the table is a large, empty rectangular area.

Figure 21: The channel information tab of the emulator.

- Sampling frequency
 - The extracted sampling frequency.
- Site info
 - The extracted site info.
- Channel index

- The indices of every specific channel extracted from the data file. This is the index that is to be specified in the **Channel selector** matrix in the **Channel Monitoring** tab.
- Receiving node
 - The node ID of the receiving node. For debugging purposes.
- Transmitting node
 - The node ID of the transmitting node. For debugging purposes.
- Channel info
 - Extracted information text. Can be used to easily identify channels.

A.2.4 Status

This tab contains information on the status of the emulator hardware. Pictured in figure 22.

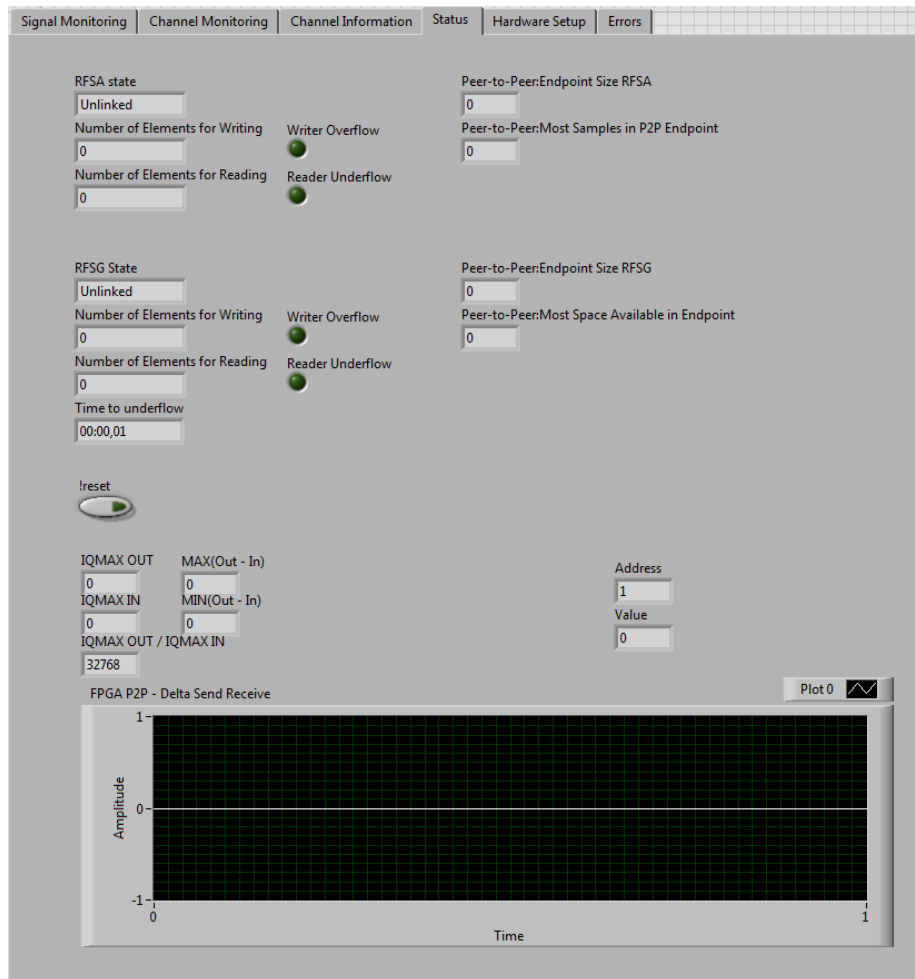


Figure 22: The status tab of the emulator.

- RFSA state
 - The state of the RFSA.
- Number of Elements for Writing
 - Elements free for writing in the P2P pipe for the RFSA and RFSG, respectively.
- Number of Elements for Reading
 - Elements available for reading from the P2P pipe for the RFSA and RFSG, respectively.

- Writer overflow
 - Indicates if an overflow has occurred in the RFSA or RFSG.
- Reader underflow
 - Indicates if an underflow has occurred in the RFSA or RFSG.
- Time to underflow
 - Indicates the running time before the first underflow occurs.
- Peer-to-Peer:Endpoint Size RFSA
 - The maximum number of samples that can fit in the P2P endpoint of the RFSA.
- Peer-to-Peer:Most Samples in P2P endpoint
 - The maximum number of samples in the P2P endpoint of the RFSA since start.
- Peer-to-Peer:Endpoint Size RFSG
 - The maximum number of samples that can fit in the P2P endpoint of the RFSG.
- Peer-to-Peer:Most Space Available in Endpoint
 - The maximum number of free space (in number of samples) in the P2P endpoint of the RFSG since start.
- Address
 - The address of the channel coefficient that is currently being updated. Should loop from 0 to 31.
- Value
 - The value that is currently being written to the indicated channel coefficient.
- FPGA P2P - Delta Send Receive
 - Plots the MAX(out-in) over time. Should remain somewhat constant over time, otherwise, a timing error is probably occurring between the RFSA and RFSG.

A.2.5 Hardware Setup

This tab contains the controls for configuring the emulator hardware. Pictured in figure B.

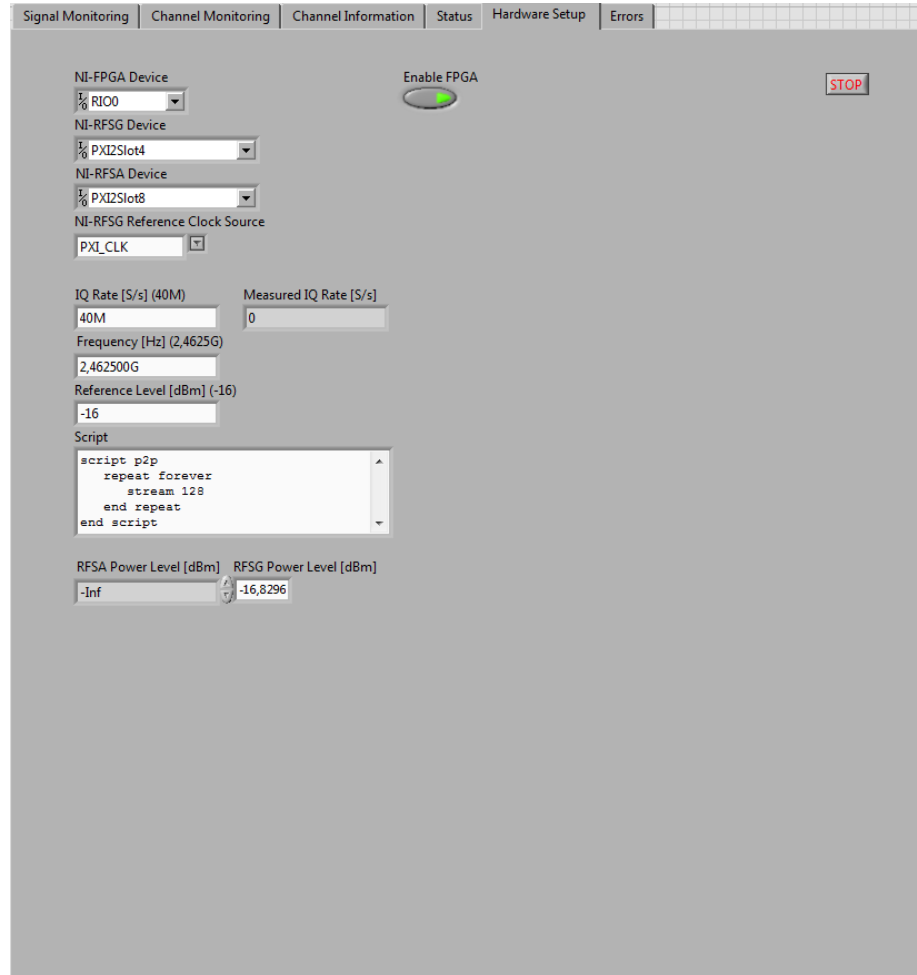


Figure 23: The hardware setup tab of the emulator.

- NI-FPGA Device
 - Specifies the NI-RIO device corresponding to the FPGA device.
- Enable FPGA
 - Enable or disable the FPGA.
- NI-RFSG Device

- Specifies the PXI-slot containing the RFSG device.
- NI-RFSA Device
 - Specifies the PXI-slot containing the RFSA device
- NI-RFSG Reference Clock Source
 - The clock to be used to control the timing of the RFSG. Must be set to PXI_CLK to avoid timing issues between the RFSA and RFSG.
- IQ Rate
 - The bandwidth sampled by the RFSA
- Measured IQ Rate
 - Measured bandwidth of the RFSG, should match the IQ rate.
- Frequency
 - Center frequency of the emulators input channels. When using the frequency channels between 2.445 GHz and 2.480 GHz, this should be set to 2.4625 GHz.
- Reference level
 - The power, in dBm, corresponding to the largest input signal that can be represented in the emulator.
- Script
 - Controls how samples should be moved over the PCIe link. May affect the emergence of underflow errors.
- RFSA Power Level
 - The power level at the RFSA.
- RFSG Power Level
 - The highest possible power generated by the RFSG.

A.2.6 Errors

This tab collects error messages produced by the emulator. Pictured in figure 24.

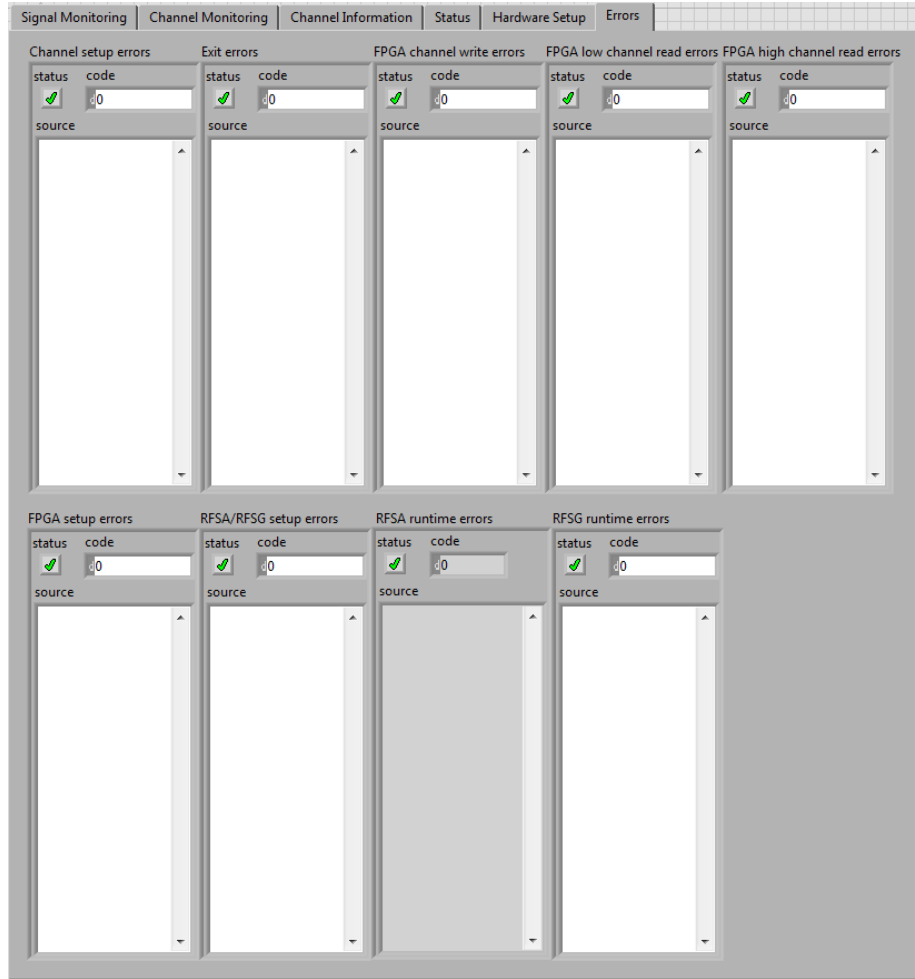


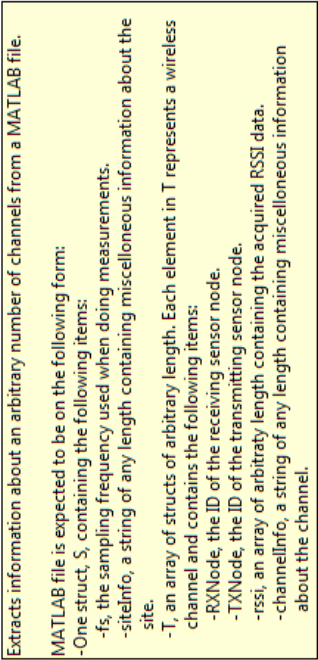
Figure 24: The errors tab of the emulator.

- Channel setup errors
 - Errors that occur while the channel coefficient matrix is being initiated.
- Exit errors
 - Errors that occur during the shutdown of the emulator hardware, i.e. shutdown of the P2P pipes, RFSA device and RFSG device.

- FPGA channel write errors
 - Errors that occur during the writing of channel data to the FPGA.
- FPGA low channel read errors
 - Errors that occur during the read of emulator output.
- FPGA high channel read errors
 - Errors that occur during the read of emulator input.
- FPGA setup errors
 - Errors that occur when the FPGA is initialized.
- RFSA/RFSG setup errors
 - Errors that occur when the RFSA and RFSG are initialized.
- RFSA run time errors
 - Errors that indicate issues with the data acquisition of the RFSA.
- RFSG run time errors
 - Errors that indicate issues with the data output of the RFSG.

B The Channel Data Import subVI

The channel data import subVI operates by iterating through the data fields discovered in the data file. Data files of an unexpected format might produce unexpected result. For readers familiar with the LabVIEW development environment, see figure 25 for details of how the subVI works.



70

C The Signal Data Export subVI

The signal data export subVI continuously writes a stream of signal data to the end of a data file. The subVI is illustrated in figure 26.

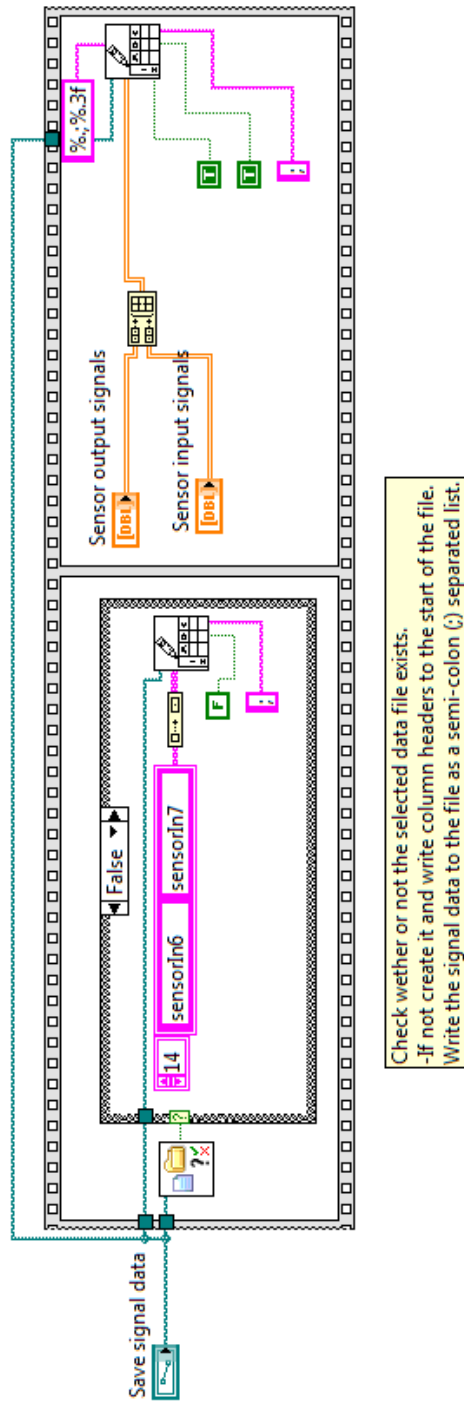


Figure 26: The signal data export subVI continuously appends data to a specified file.